THE UNIVERSITY OF ALBERTA

SYSTEM DESIGN USING A [illegible] COMPUTER

by

© J. Barry [illegible]

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

Spring, 1971

THE UNIVERSITY OF ALBERTA

SYSTEM STUDIES USING A SIMULATED COMPUTER

by

© J. Barry Dutton

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

Spring, 1971

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled SYSTEM STUDIES USING A SIMULATED COMPUTER submitted by J. Barry Dutton in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

As the study, at the undergraduate level, of hardware/software systems becomes increasingly important in university curricula, the need for adequate facilities to process the whole range of system oriented assignments becomes critical. Neither the manufacturer's software nor the traditional assembler language processors for student jobs completely fulfill the need. The ideal facility would be to supply each student with an actual computer, complete with software designed to be replaced in stages with student developed software. Since this approach is not economically feasible, a language processor for student programs, the Student Assembler Language Translator (SALT), has been extended to simulate efficiently to a student the characteristics of a simplified /360 computer (SUPERSALT) complete with a replaceable supervisor (SALT MONITOR). How this extended processor might be used is explored in the outline, including assignments to be run on the simulator, of an undergraduate systems course.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my supervisor, Dr. J.P. Penny, for his guidance and encouragement in the preparation of this thesis; to Dr. B. Mailloux for his helpful suggestions and continued interest in the project; and to my wife for her many hours of patient work in typing and proofreading the manuscript.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Undergraduate Education in Computer Systems

A major objective in the education of computer scientists as opposed to computer users, should be to reduce to a minimum the lack of understanding of computer systems, both software and hardware. Unfortunately it is often true that even skilled users and competent theoretic computer scientists lack comprehension of the principles governing system design. Naturally, such a shortcoming results in an inability to exploit fully the available computing facilities and, potentially more serious, an inability to communicate with those responsible for system design, maintenance and operation. The author has found, for example, that even graduate students, for whom systems studies offer an absorbing field, are often more at ease with the theoretical literature than with the actual systems at their disposal.

During their careers, Computer Science graduates may be expected to engage in such activities as systems programming, consulting, computer management and software engineering, all of which require more than an exclusively theoretical background. They may need to participate in the process of hardware/software selection and thus find it essential to be capable of translating the characteristics of various configurations into terms of operating efficiencies, resource loads and possible bottleneck areas.

Computer Science curricula now appear to be deficient in this

important area of systems studies. An examination of such curricula for universities in Britain [3] and in Canada [1] shows that topics such as assembler language, logic design and language processor construction are usually included. On the other hand, material on computer system structures is frequently absent from curricula and, where taught at all, may be covered only in the final year of the program. Admittedly, the surveys quoted are now three years old, and the situation is certainly changing. The most recent ACM curriculum recommendations [2] include three courses, two intermediate and one advanced, whose synopses show a good coverage of systems topics. However, personal observation suggests that very few universities even now offer such a variety of courses.

Even in those which do, the value of discussing theoretical system concepts is diminished by the lack of opportunity to gain practical experience. Similarly, the teaching of a programming language loses impact if programs cannot be run. The manufacturer's software offers one possible vehicle for gaining a systems background but from the author's experience, it also suffers serious drawbacks. Special student job processors for low level languages are common and might be used as the basis for assignments illustrating system principles, but these have most often been developed to minimize interference with other processing, rather than with a view to providing comprehensive facilities for experimentation. The author was earlier associated with the development of such a system, the Students' Assembler Language Translator (Dutton [7]; Easton and Penny [10]), which is an in-core system for processing student assembler language

jobs on an IBM /360. The original aim was efficiency, but an extended version, designed and implemented as part of this thesis project, has been developed to provide a more complete and realistic simulation of a computer system.

In this thesis we are concerned with:

(a) exploring the shortcomings and inconsistencies of some current student assembler language processors, particularly SALT, as illustrative tools for systems courses,

(b) specifying the characteristics of a /360-like pseudo computer that retains the mechanisms fundamental to this genre of hardware while eliminating the non-illustrative detail that makes creating software particularly time consuming and difficult,

(c) describing some of the main problems encountered in presenting a systems course using the manufacturer's software both as an example for illustration and as the framework within which assignments were done,

(d) outlining how a similar systems course could have been structured around a project using a comprehensive student system such as that described in (b), had it been available,

(e) explaining the major problems in extending the SALT system from a language processor with the shortcomings discussed in (a) to a simulator of the computer system specified in (b).

CHAPTER II

STUDENT PROCESSORS: THE PROBLEM

2.1 Introduction

An examination of the student oriented assembler processing systems, for which the author has been able to acquire documentation, reveals that although they provide an excellent simulation of a low level language and certain aspects of the internal machine architecture, they are incomplete or inaccurate in their illustration of several fundamental machine characteristics. The authors of SOS: The Brown University Student Operating System [4] pinpoint a major problem area by indicating that the SOS 'machine' does not overlap its I/O with processing but, on the other hand, they have retained for pedagogical purposes the concept of the data channel. It is not clear what pedagogical purpose is served by a channel incapable of asynchronous operation witl the CPU, since its existence thus seems pointless.

The other major problem area is the absence of an explicitly defined interrupt mechanism in the simulated processors. This mechanism is completely absent in all three student systems: SOS, STASS and SALT. The following discussion will explore in more depth the problems as they occur in SALT, with the understanding that a similar discussion could be made of either SOS or STASS.

2.2 The Inconsistent Time Base

Consider the following program segment:

```
REPEAT  LA    R3,5(R3)
        LA    5,10(5)
        MVC   0(20,3),0(5)
        CR    5,3
        BH    REPEAT
        .
        .
        .
```

No inconsistencies, for the purpose of illustration, result from the execution of these instructions on the hypothetical SALT computer. The memory organization and addressing scheme are faithfully reflected and the CPU procedure for calculating an address is easily described. The existence and nature of micro-programs can be readily illustrated with reference to the machine instruction formats. The existence of general purpose registers is apparent, and special purpose registers and conditional branching control to explain in detail the inner workings of a central processor, are easily postulated. Most important, the time base is realistic since during interpretive execution of the instructions, accurate execution times (for a /360 Model 65) are calculated by the interpreter for each instruction (Fig. 2-1).

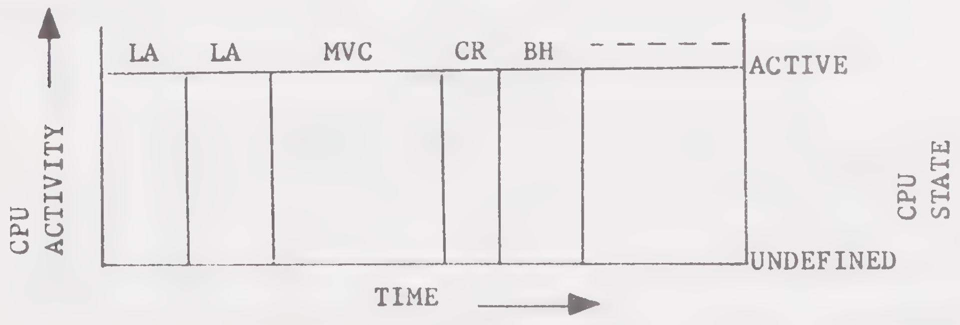


Fig. 2-1. CPU status during uninterrupted execution

However, consider a program segment containing an I/O request:

```
               LA    5,10(5)
               READ  CARD
     +         LA    1,CARD
     +         SVC   1
               MVC   0(4,5),CARD+20
                .
                .
                .
          CARD DS    80C
```

The I/O request is realistically interfaced to the SALT hardware through a system macro "READ" which places the input buffer address in register 1 and generates a supervisor call to the SALT monitor.

The activity performed by the monitor is, however, impossible to describe precisely since the instruction following the supervisor call appears to be executed without an intervening pause and may, in fact, refer to the information just obtained in the READ operation. That is, the record inexplicably appears in memory in the instant between the execution of the SVC and the following instruction. This discontinuity in the time base is illustrated in Fig. 2-2.

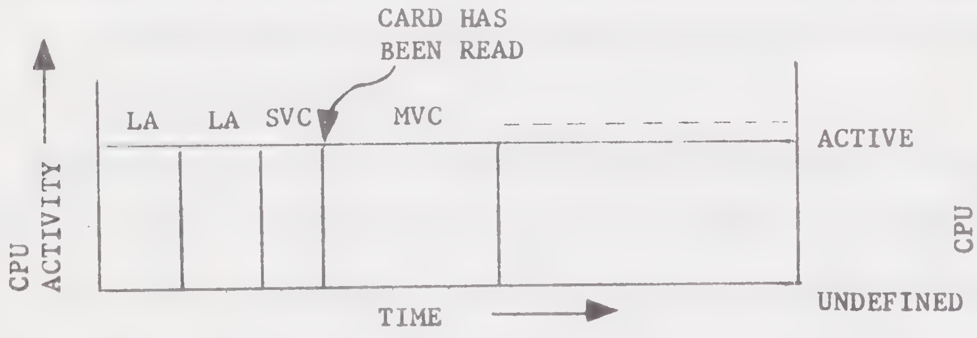


Fig. 2-2. CPU status does not demonstrate interrupted execution

Besides the inconsistent time base, SALT has several other shortcomings which severely limit its usefulness for illustrating the principles of hardware/software systems. These are:

(a) Communication between the CPU, in which SALT instructions appear to be executed, and its environment is not well defined to the user since I/O devices are connected, and operate in an invisible fashion.

(b) The characteristics that can be ascribed to the SALT CPU from examination of SALT machine instruction execution in SALT time indicate that the CPU has a single state, i.e., running.

(c) Even if other hypothetical states are postulated for the CPU, there is no hardware mechanism, defined and visible to the student, for changing the CPU state, for terminating or switching the execution of instructions from a given instruction stream, or for initiating CPU operation at some time T.

(d) Even though the existence of software support for the student's program is demonstrable, and the existence of hardware is obvious, it is not at all clear precisely where hardware ends and software begins or how the one is interfaced to the other.

Appendices I to IV list detailed specifications for a minimal, /360-like machine, SUPERSALT, with software support such that:

(a) Pseudo programmable channels with pseudo devices attached are visible to the user, and these channels appear to oper-

ate asynchronously with the SUPERSALT CPU in a fashion consistent with the simulated time base.

(b) A multi-state CPU with precisely defined hardware characteristics is apparent and available to the programmer, either as a bare machine on which user-supplied software can be employed to run user programs, or in combination with the SALT monitor employed to run user-supplied programs, or in virtually any combination of SALT monitor and user software support.

The following program segment, using the facilities described in Appendices III and IV, would, when executed on SUPERSALT under the SALT monitor, appear to the user as having a fully consistent time base (Fig. 2-3).

```
         LA    5,10(5)
         EXCP  CRDRDR
+        L     1,=A(CRDRDR)
+        SVC   6
         MVC   0(4,5),0(3)
         LA    3,10(3)
         WAIT  CRDRDR
+        L     1,=A(CRDRDR)
+        TM    10(1),X'01'
+        BO    *+6
+        SVC   7
         L     5,0(3)
         .
         .
         .
```

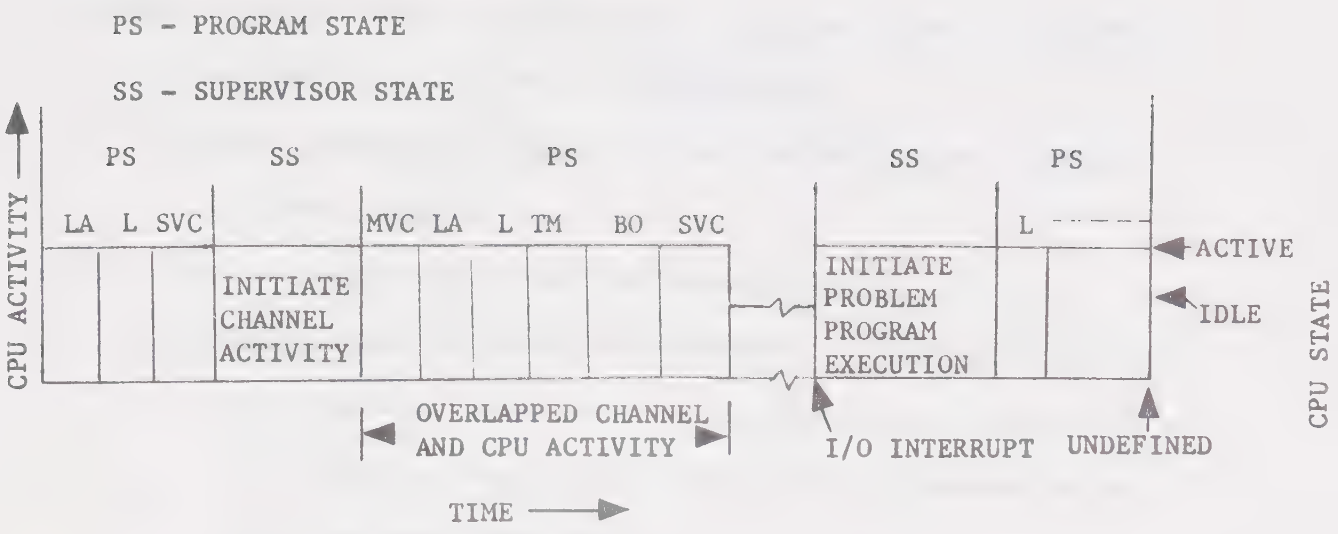


Fig. 2-3. CPU status reflects the synchronization of events occurring within the SUPERSALT computer

Before specifying the characteristics of a student-oriented processor suitable for running students' problems in a systems course, it is first necessary to make some assumptions concerning which aspects of hardware/software systems might be required for student use. The author believes that the most useful facility for illustrating systems concepts would be a set of hardware (simulated or real), complete with software which the student could replace with his own. Since actual hardware is not normally available for the exclusive use of individual students, and since the already existing SALT system offered an excellent base on which to build a simulator, the author decided to proceed from that point. This decision restricted the simulated hardware to the possession of characteristics somewhat like those of an IBM /360 computer. However, in those instances where the author had the opportunity to influence the characteristics of the hardware/software combin-

ation, he did so with the intention of creating a tool of suitable power and complexity for use by students who:

(a) had completed at least one, and preferably two, Computing Science courses,

(b) had mastered an assembler language, and

(c) had reached the level of development at which they had become somewhat blasé about programming algorithms, and were interested in learning more about the environment in which their programs were run.

They could then choose to take as their first course in computer systems, one in which the following areas are explored:

(a) CPU characteristics:

  (i) the nature of instruction execution by the CPU,

  (ii) the need for multiple CPU states to control instruction execution and the instruction set recognized by the CPU at any given time,

  (iii) the concept of instantaneous CPU status,

  (iv) the need for a mechanism, both to save the instantaneous CPU status and to reset it to new, predetermined values in order to change CPU states, and to switch the instruction stream being executed.

(b) Channel characteristics and channel/CPU interaction:

  (i) the nature of a channel as a primitive processor and the reasons for making it programmable,

  (ii) the concept of multi-processing as exemplified by asynchronous, overlapped operation of channel and CPU,

(iii) the need for making the channel capable of interrupting CPU instruction execution to facilitate synchronization and control of I/O and program operation.

(c) Software:

(i) software as the interface between the hardware and the program,

(ii) the concept of continuous, protected operation through software management,

(iii) the use of software to resolve speed mismatches between hardware components ,

(iv) software systems for efficient machine utilization: the concept of "task" and the synchronization constraints between tasks; multiprogramming within job bounds and outside job bounds,

(v) introduction to data management: external storage; file organization,

(vi) storage management and memory protection,

(vii) the effects of device characteristics and system configurations on system tuning,

(viii) initiating processing on the hardware/software combination.

CHAPTER III

SPECIFICATION OF A SIMULATED COMPUTER SYSTEM

3.1 Introduction

An extended SALT system, simulating a computer called SUPERSALT, has been designed to provide students with facilities normally present in an actual hardware/software system, but not usually available or visible to the programmer. The student is given the facility to specify within each job to be run on SUPERSALT:

(a) a configuration of peripherals and their characteristics,

(b) whether the SALT monitor is to reside in SUPERSALT or whether the job contains the user's own monitor to supervise the execution of his program.

As well as specifying the hardware configuration, the user may program the peripheral devices from his program and have the execution of these channel programs overlap the execution of his SALT instruction. If the user elects to include, as his software configuration, his own monitor, then he essentially has complete control of the simulated hardware including the handling and masking of interrupts and the testing and initiation of channel operation.

The SALT system, however, maintains control over the transition between jobs, and prints out statistical and diagnostic information as required.

3.2 Specifying the Hardware

The peripheral configuration is specified on the $SALT card in

the format shown in Appendix II. Currently the peripherals are restricted to pre-set, fixed length records, read or written in a strict physical sequence that is not re-transmittable, i.e., a record once read cannot be re-read. In these characteristics, the peripherals are similar to card readers or line printers, but they differ in that the average rate of delay before transmission (the time required to physically read or write the record at the device) and the data transmission rate (the rate at which the data is transmitted across the channel) are specified by the programmer. These reader/printer-like devices and their channels can, through changing their characteristics, be made to operate at rates varying from typical unit record values up to main memory speed, thus demonstrating CPU usage as a function of CPU/peripheral speed mismatch. The characteristics of a channel other than the channel address or data rate are pre-set. A channel can have more than one device attached to it but it can service (execute a channel program for) only one device at a time. All channels have data widths of 160 bits, i.e., they are capable of transmitting 20 bytes in parallel. These 160 bit wide transmissions follow one another serially at intervals of (channel width × constant)/(data transfer rate) seconds.

3.3 Specifying the Software

The software configuration also is specified on the $SALT card through the presence or absence of the relative address of a reserved memory location in the program into which the SUPERSALT hardware can exchange status information. If the relative address is absent or

invalid, then the status exchange mechanism is not visible to the programmer, and all requests for supervisory functions are serviced by the SALT monitor. If the relative address is present in the format specified in Appendix II, then the hardware functions of status exchange, initiation of channel operation, timer setting and memory protection bound definition are placed under user control through the values in the specified reserved core area. The format of this reserved memory is shown in Appendix II. Programmer control of these hardware functions normally implies that a user monitor forming one part of a SALT job will service supervisor requests from a problem program which forms another part of the same job.

The extended system, however, maintains overall control over individual jobs by monitoring execution time and pages of output, and overseeing job-to-job transition. The system may be called upon to perform supervisory functions which the programmer chose not to or could not accommodate in his own monitor.

3.4 <u>Performing Input/Output</u>

If the "DEVICE=" option was specified on the $SALT card then the user may include, as part of his SALT job, channel programs for the channels connecting the devices to the CPU, and he may use SALT system macros to:

(a) cause initiation of individual channel programs,

(b) relate channel programs to a given channel and device,

(c) impose those synchronization constraints between channel

operation and instruction execution that are required by the logic of his program.

The characteristics of the channel command words (CCW's) that make up channel programs are presented in Appendix III and the system macros used to control channel execution are described in Appendix IV.

SUPERSALT channels, once activated, behave as elementary computer processors. They execute, in place of instructions, CCW's retrieved sequentially from main memory by the channel. By executing a special instruction within the CPU a channel is directed to fetch the first CCW of a channel program (a set of logically connected CCW's). However, after activation, the channel appears to fetch and execute CCW's asynchronously with the CPU execution of instructions. Of the four CCW commands, only two (READ and WRITE) can cause data to be transferred across the channel. Since currently supported SUPERSALT devices are considered as having their own internal buffer storage, the reading or writing of records is not concurrent with the transmission of data across the channel. The execution of a CCW READ command causes the record information to be inserted into the reader's internal buffer from where it is transferred down the channel in amounts determined by the channel program. Similarly, on output, information is transferred under control of a channel program from main memory to part or all of the buffer storage in the output device, after which the entire contents of the buffer become the output record. Thus a single CCW cannot cause more than one record to be written or read, but a

single record may be written or read by a number of CCW's chained together (see Chain Data in Appendix III). Similarly, another type of chaining may cause several records to be read or written by a string of CCW's forming a single channel program requiring only one initiation by the CPU (see Chain Command in Appendix III).

Two other commands can be issued in a CCW. The CONTROL command causes mechanical activity such as spacing or skipping on the printer. The TRANSFER IN CHANNEL command causes no activity at a device, but instead, it instructs the channel to fetch a new CCW, not the one in the next contiguous eight bytes as would happen with chained CCW's, but rather the eight bytes starting at the address specified in the "addr" field of the TRANSFER CCW. It is analogous to an unconditional branch instruction.

A great deal of care was taken to ensure that I/O activity appears to take place along a realistic but uncluttered time base. A typical sequence of events for a READ operation might be:

(a) CPU execution of a load instruction (part of the expansion of an EXCP macro) places the address of a CCB in register 1.

(b) CPU execution of the EXCP SVC instruction results in a SUPERSALT SVC interrupt.

(c) If the user has provided his own supervisor as is discussed in Section 3.5, the next event is the CPU execution of the instruction at the address specified in the new SVC PSW,

(d) If no user supervisor has been provided (which is the situation assumed in (e) to (h) below), the SALT monitor services

the SVC by initiating, if possible, the execution of the channel program.

(e) At a SALT time of 100 μsec after the execution of the SVC, control returns to the user-supplied instruction which follows the EXCP SVC. (This is the time required by the supervisor to initiate I/O.)

(f) Instruction execution continues until either a WAIT SVC is executed or a time equal to the specified average delay time for the device has passed. In the latter case twenty bytes of the input record under CCW control are placed into memory.

(g) Instruction execution continues with subsequent twenty byte portions of the input record being stored according to CCW specifications at intervals of (20/data transmission rate) × (constant) μsec. This continues until either the channel program terminates or a WAIT SVC is executed.

(h) If a WAIT SVC is executed, the CPU appears to be placed in the WAIT state for a length of time equal to that required to complete the channel program, after which control returns to the instruction following the WAIT SVC.

The system macros of Appendix IV will not execute properly unless the "DEVICE=" option has been entered on the $SALT card, and the devices and channels have the same addresses as those specified in the Channel Control Blocks. More than one device may be attached to a single channel but, since a channel is wholly dedicated to a device during the time when a channel program is being executed, any

attempt to perform I/O on the other device will cause abnormal termination of the job (if run under the SALT monitor). That is, it is the programmer's responsibility while using these system macros under the SALT monitor to synchronize his I/O requests in order to avoid resource contention, since the SALT monitor does not queue multiple requests for the same resource. It is, however, quite possible to have separate channels controlling separate devices, operating simultaneously with CPU execution of SALT instructions.

If the "DEVICE=" option is not present on the $SALT card, the sole means of performing I/O becomes the original READ/WRITE system macros described by Penny and Easton [10]. Since no channel or device address is associated with these macros, they must be considered as performing I/O on the "system input" or "system output" devices.

If the "DEVICE=" option is present, the READ/WRITE macros can still be executed; however, no CPU channel overlap results. For example, the execution of a READ would cause a record to be read on the unit specified in the "DEVICE=READER,..." field (if one is present) on the $SALT card. Control would be returned to the instruction following the READ SVC only after the entire record had been transmitted to the address specified in the READ. When a READ or WRITE is executed, care must be taken to ensure that no channel program, initiated by an EXCP macro, is in execution on the required device since the resulting I/O will be a combination of the records. The READ and WRITE can be considered a higher level of logical I/O support than the EXCP.

Examples of channel programs and the corresponding system

macros appear in Appendix V. A detailed description of channel programming appears in Dutton [5].

3.5 Replacing the SALT Monitor

If the "PSW=" option appears on the $SALT statement the SALT program, once assembled, is treated as a stand-alone program running on a "bare" machine. Although system software support is still present to take over in case the SALT program cannot maintain execution on the bare machine or in case convenience services are requested by the executing program, this underlying level of software is kept invisible as much as possible. The SUPERSALT machine is defined to be consistent and self-contained within the limits of execution of one SALT job. That is, it has:

(a) a visible interrupt system to which the user must interface his SALT program,

(b) a program activated supervisor state in which the CPU recognizes special instructions,

(c) a CPU that is either executing (or prevented from executing) along a consistent time base,

(d) provision for a memory protection mechanism (not currently implemented),

(e) provision for a timer to be decremented in SALT time (not currently implemented).

The exact relationship between the user software on the "bare" machine and the SALT monitor is that any program or SVC interrupt occurring when the SUPERSALT CPU is in the supervisor state is

intercepted by the monitor without reference to the user PSW's. Conceptually this is undesirable since the SALT monitor appears to run on an undefined machine, but practically it is both a necessity to maintain the flow of SALT jobs (by placing limits on execution time, lines printed and level of interrupts) and a considerable convenience to the user since he can use the facilities of the monitor during development of his own software by reflecting selected interrupts to it. In any case, since there are no instructions in SUPERSALT that operate on packed decimal fields, the process of converting from EBCDIC to binary and vice versa should be reflected to the monitor, and since job-to-job transition is beyond the user's control, the normal end of SALT job must be accomplished through the monitor.

An example program containing user PSW's and a simple set of interrupt handlers can be found in Appendix V. A more detailed description of interrupts and PSW's appears in Dutton [5].

CHAPTER IV

THE SIMULATED COMPUTER AS AN AID IN PRESENTING A COMPUTER SYSTEMS COURSE

4.1 Introduction

The system specified in the previous chapter might be used to advantage when setting assignments in a course dealing with software/hardware systems. The course would most naturally follow those in which the student had been exposed to high level languages and to some Assembler Language, and might form a third course in Computing Science. The facilities offered by the extension appear to provide a solution to the problems encountered by the author in setting assignments for such a course, and also appear to overcome the shortcomings, for purposes of instruction, of the manufacturer's hardware/software system. The following section, 4.2, describes some of these problems experienced by the author when teaching a third-year course in systems, using only the manufacturer's software. Section 4.3 outlines a hypothetical, long term, modular project using SUPERSALT as the basis for the assignments in a somewhat similar course.

4.2 Using the Manufacturer's Software

During the first term of the 1970-1971 academic year at the University of Alberta, the author was responsible for organizing and presenting a third-year course consisting of the study of a particular operating system - in this case, the IBM /360 Operating

System. The approach taken was to study the system from three different points of view:

(a) that of the applications programmer,

(b) that of the systems programmer, and

(c) that of the system designer.

Before examining the system structure in point (c), the class was presented with a description of the characteristics of the /360 CPU and the relationship between channel and CPU operation. It was felt that no explicit understanding of the characteristics and operation of the software could be possible without a basic knowledge of the hardware to which the system was designed to interface.

A set of five programming assignments using the manufacturer's software under /360 OS-MVT was designed to provide, as far as possible, practical illustrations of the more important aspects of the lecture material. These assignments were constructed in such a way as to build upon each other and to emphasize in succession:

(a) the system command language (JCL),

(b) the Assembler Language features,

(c) the data set organization and several levels of data management facilities,

(d) the system utilities and external storage organization,

(e) the system macros,

(f) the dynamic program structure and multi-tasking.

An examination of these points reveals that the first three illustrate some aspects of the operating system from the programmer's point of view, and probably all six are of concern to the systems

programmer; however none of them provides a detailed insight into hardware interactions, and none can be considered particularly illustrative of the internal structure of the software. Perhaps the last assignment provided a slight indication of systems operation in that it consisted of a main program that ATTACH'ed three asynchronous subtasks, one to read the input data, while another processed the card image and a third printed out the results. This organization slightly resembles the spooling function required in a multi-programming environment. In fact, during the latter part of the course it became necessary to explain, without student opportunity for practical observation, the other side of the picture, i.e., what was required in the way of system software to "accommodate" the students' programs on the hardware. It does not appear feasible to illustrate through programming assignments using the /360 OS-MVT software, this most important aspect of systems study (short of allowing each student access to the actual hardware).

Another problem that became quite perplexing toward the end of the course was how to determine the level of detail to present in the lectures. During the first part of the course which dealt with programming and systems programming, it was quite feasible and desirable to conduct the lectures on a general plane. The student was expected to fill in the details through massive reading assignments using the manufacturer's manuals. However, during the latter part of the course when the emphasis shifted toward studying the internals of the system, the lectures became almost the sole source of information since the manufacturer's Operating System Program Logic Manuals

were hopelessly detailed, and yet they formed the only comprehensive description of the system structure that the author had encountered. This made it difficult to attain an appropriate level of complexity. The following approaches were considered:

(a) formulation of the system as an idealized abstraction,

(b) presentation of the system by describing its control blocks, queues, etc.,

(c) presentation of a combination of the above, a conceptual description supported by a somewhat simplified description of the main control blocks and other system components.

The first alternative was rejected since it contravened the original aim of the course, to provide a pragmatic view of an actual operating system. The second was not seriously considered since there was insufficient time to present even a small fraction of the bulk of detail involved, and much of it would have been irrelevant to a "functional" understanding of the system in any case. The third alternative was chosen and found to be feasible, although far from ideal. The author found it disconcertingly easy to eliminate control blocks and other components thought to be superfluous to the understanding of the functioning system, only to find they (or some part of them) were essential in explaining some other characteristic of the software. Also, when a system function was reduced to its essentials, it must have seemed confusing to the student who felt he understood the function, to later encounter in his outside reading other seemingly unnecessary components.

A third problem, one that appears to plague any course which

uses the more exotic facilities of a software system, was the high risk of causing malfunction of the resident software through student programming errors, thus interrupting normal service to all users. No amount of precaution can completely eliminate this risk. In an attempt to pinpoint sensitive areas, all five of the assignments for the course were programmed before the course began. When each assignment was handed out, a special workshop session was held, in which the original programmer of the problem discussed these areas and suggested how the problem might be programmed. In addition, each student program was run under a small monitor in an attempt to isolate that program from the system to some small degree.

In spite of these precautions, a few systems failures were traced definitely to the students' programs, and several more failures were suspected to have been caused by them. In any event, some disturbance to normal system operation was experienced, although not to such a degree as to cause revision of the assignment plans.

A final factor that must be considered for any course in which programming forms the bulk of the assignments is the cost of resources required from the computer installation for the debugging and running of student programs. In total, the five assignments mentioned above required approximately:

(a) 10 cylinders of 2314 disk space reserved over a period of two months,

(b) 137 computer runs (separate OS jobs, each requiring 100 K of memory) per student,

(c) 80 minutes execution time on a /360 Model 65 per student,

for 23 students. It should be emphasized that the figures in (b) and (c) above were obtained by extrapolation of incomplete data. However, even if these figures are taken only as a very rough guide, they indicate a considerable investment of resources for a single half-course.

4.3 Using Special Student Oriented Software

The SALT system described in Chapter III could be used to support a set of assignments around which a system course, similar to that discussed in the previous section but without the attendant problems, could be built. Each assignment is designed to build on those preceding it as did the set described in 4.2, but the intention in this case is to develop in parallel both a user program of progressively greater complexity and the supervisor functions necessary to support it. In its final form, the student will have created a full supervisor under which runs an assembler of his own creation, which uses supervisor functions to assemble and load programs which in turn use supervisor functions during their execution. The lecture material which these assignments are designed to illustrate would cover much the same ground as the course described in Section 4.2 but the manner of exposition would be radically different. In the first case, an attempt was made to simplify (and generalize from) a currently existing operating system of great complexity as an example of a typical operating system. In the other case, the lectures would attempt, given the SUPERSALT hardware characteristics, to explore a variety of software configurations as solutions to varying

design aims.

The first assignment (Fig. 4-1) is intended to demonstrate modularity in a SALT program and the use of standard linkage conventions such as those used in OS, as the interface mechanism between modules. The assignment uses standard SALT system macros and the familiar SALT monitor to provide supervisor facilities. The logic of the modules need not be extensive since the important factor in the assignment is the overall structure.

While the assignment is in progress the lectures could most profitably be centered on a description of the SUPERSALT hardware under the following topics:

(a) machine instruction format,

(b) instruction fetching and separation into component fields,

(c) address generation,

(d) instruction address register (IAR),

(e) memory address registers (MAR),

(f) current CPU status,

(g) the interrupt - an exchange of status information,

(h) the program status word (PSW),

(i) memory protection mechanism,

(j) problem/supervisor state,

(k) wait/run state,

(l) interrupt classes.

At this point, an introduction to software might include a discussion of the relationship between assembler language and machine instructions.

Assignment #2 (Fig. 4-2) is intended to make use of the lecture

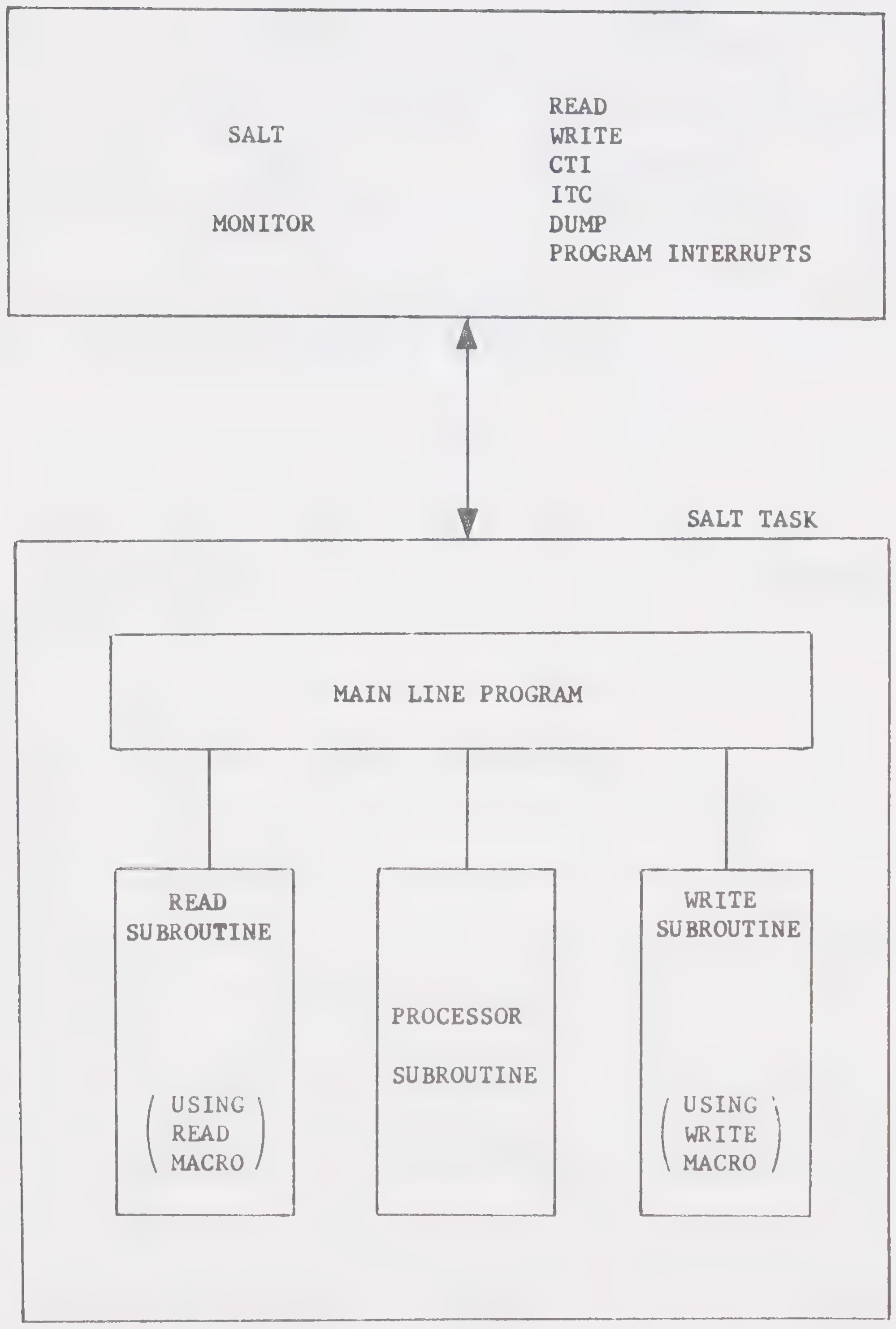


Fig. 4-1. First stage of the assignment demonstrating conventional subroutine linkage

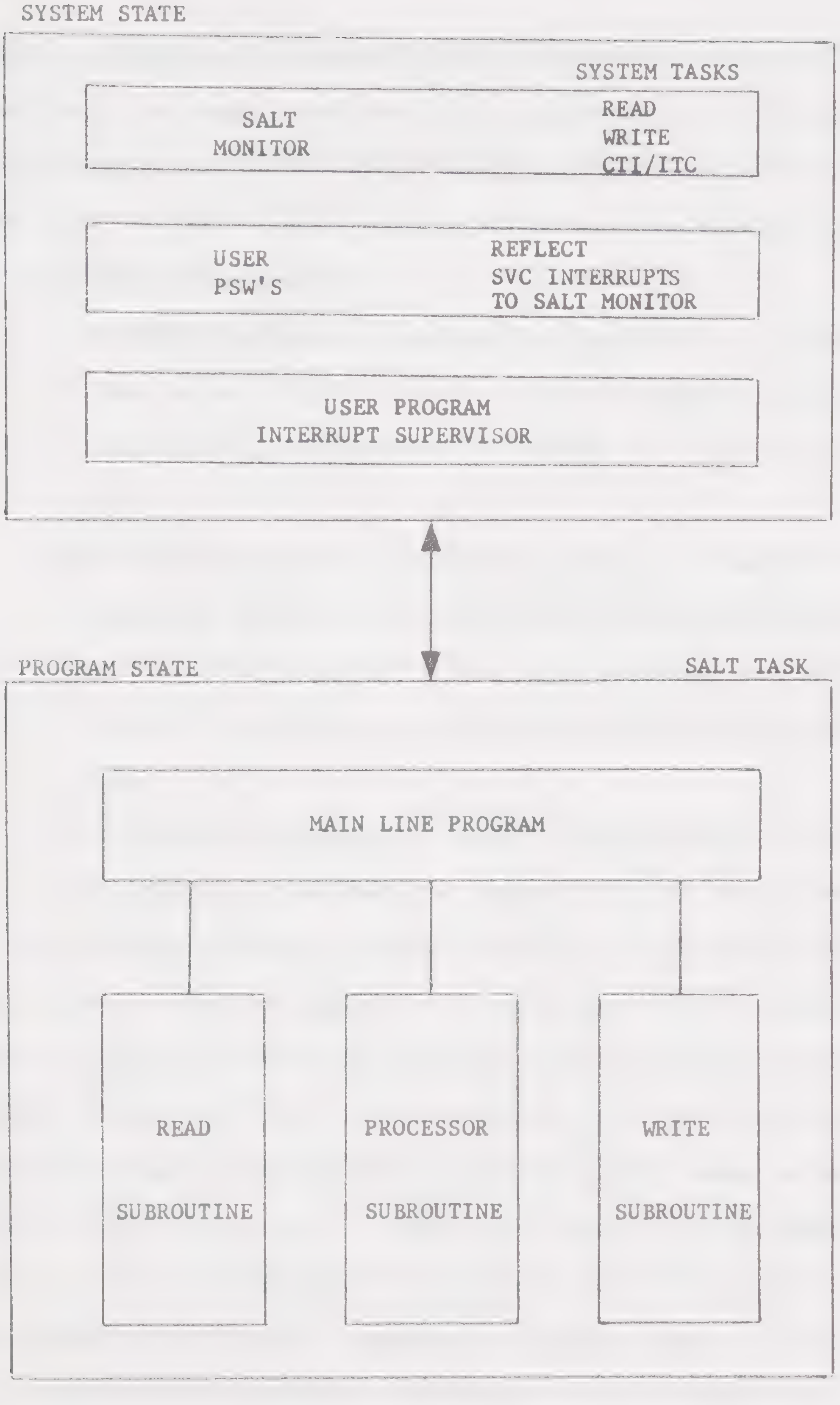


Fig. 4-2. Second stage with user program status words and user program interrupt supervisor

material on SUPERSALT hardware in that the student is required to include, with his program from the first assignment, a set of program status words at the address specified in the "PSW=" option on the $SALT card. The new PSW's specify user interrupt handling routines with the following characteristics:

(a) For SVC interrupts: a routine that executes, in the supervisor state, an SVC instruction with the same SVC code as the original, thus passing the request on to the SALT monitor.

(b) For I/O interrupts: a routine that issues an "impossible condition" message since only READ and WRITE macros have been used and their SVC's have to be reflected to the SALT monitor; thus no I/O interrupts should be visible to the user.

(c) For program interrupts: a routine that performs, for the user, the debugging function of a program interrupt supervisor.

The program interrupt supervisor can be as comprehensive as the user wishes to make it. Typically it would issue error messages and dump the register contents and selected portions of memory just as the SALT monitor would have done. The DUMP system macro would be a convenient means of accomplishing this since its SVC would be reflected to the SALT monitor. A more sophisticated version might attempt to continue execution by terminating the offending subroutine, reloading the caller's registers by following back the save area links and then continuing processing to the next interrupt. In any case an inventive user should be able to decrease the debugging

time for subsequent assignments by obtaining more information from each run.

The lecture material, while assignment #2 is being completed, would center on the following topics:

(a) channel operation and device characteristics,

(b) the channel command word and channel programs,

(c) the system macros EXCP, WAIT, CCB,

(d) overlapped I/O and buffer management,

(e) logical and physical level I/O.

Assignment #3 (Fig. 4-3) is intended to use this information concerning channels and devices to demonstrate what is involved in supporting a logical level of I/O for the WRITE subroutine of assignment #2 and to replace the user's READ subroutine with one containing channel programs.

In assignment #2 the major change to the user's supervisor is the modification of the SVC primary interrupt handling routine to recognize and intercept WRITE SVC's rather than passing them on to the SALT monitor. Interception of the WRITE request involves supplying the supervisor routine to support the following characteristics of the logical I/O:

(a) The WRITE macro specifies no device address and is assumed to request output on the system output device of a 120 byte record with carriage control.

(b) When control returns to the instruction following the SVC, the record is assumed to have been written since there is no overlap between logical I/O and instructions in the

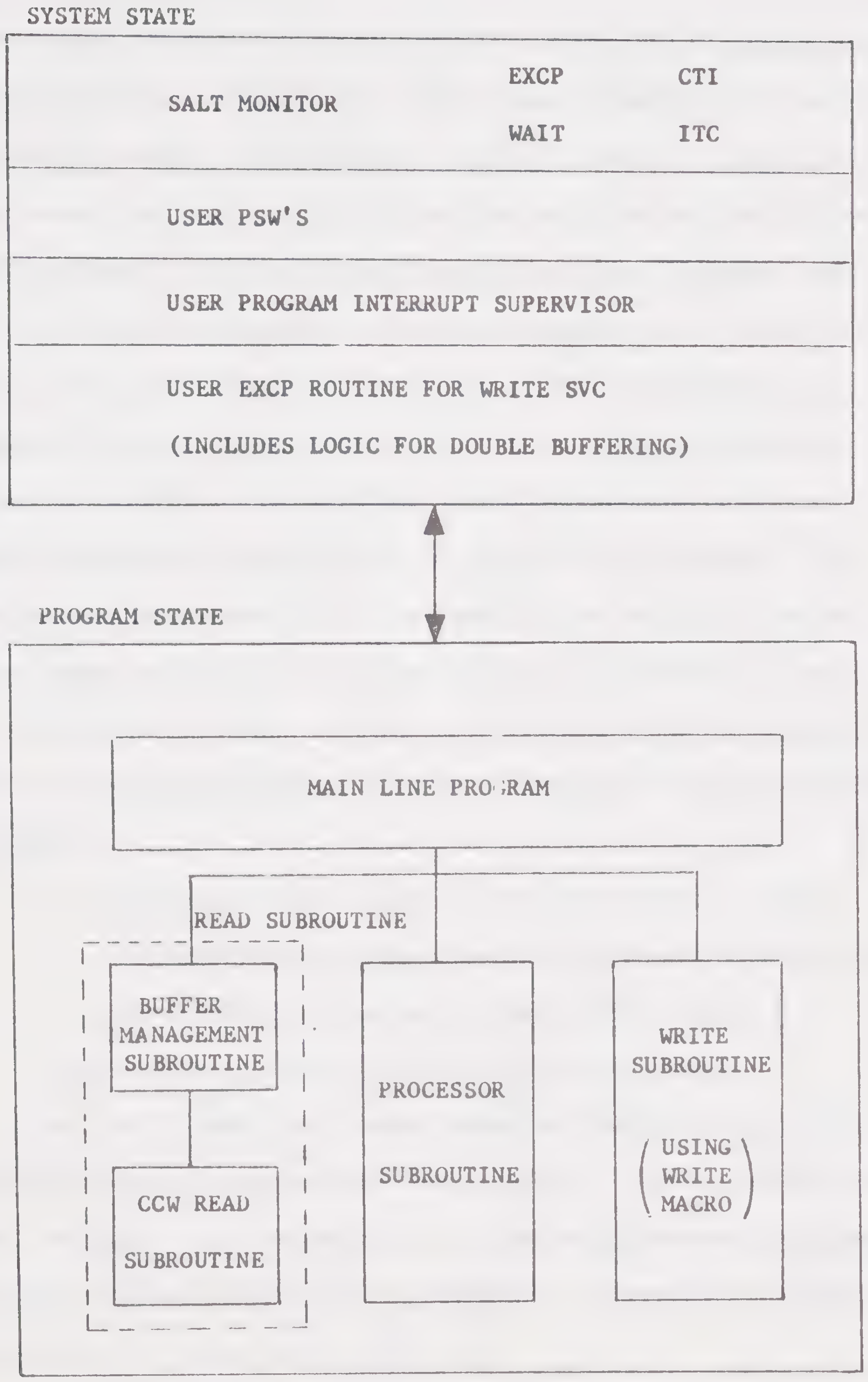


Fig. 4-3. Third stage with user CCW's and multiple buffers in place of the READ macro and user supervisor routine for the WRITE macro SVC

issuing program.

The support routine would use the EXCP, CCB and WAIT group of macros to control the execution of the channel program equivalent to the WRITE request. The EXCP and WAIT SVC's would be issued in the supervisor state and therefore they would be serviced by the SALT monitor. In order to provide some overlap of channel and CPU activity, the support routine would manage two internal buffers filled from the program I/O area by MVC's and emptied by channel programs, employing the strategy of keeping them empty whenever possible. The WAIT SVC should be executed only when a WRITE request has been made and both buffers are occupied, one buffer in the process of being emptied by the currently executing channel program and the other waiting to be emptied as soon as the channel is free. The WAIT, of course, would be issued on the currently active CCB so that when execution resumed following the WAIT:

(a) the other channel program could be initiated by EXCP,

(b) the current WRITE request could be serviced by moving the output record to the newly freed buffers, and

(c) execution of the problem program could resume.

The user's supervisor should intercept the problem program EOJ SVC in order to issue WAIT macros on each CCB in the WRITE support routine. This ensures that all the output records have been written from the buffers before the SVC is reissued in the supervisor state to cause end of SALT job.

While the WRITE support routine as part of the software system

must be designed to supply a general function, making no assumptions on the characteristics of the programs it is to service, the user's new READ routine, as part of a problem program, can be tailored to the needs of that program. This dual approach to the problem of I/O is intended to illustrate the fundamental difference between the design of systems, as opposed to application, routines.

The READ routine would contain the channel programs corresponding to perhaps three or four buffers, together with the buffer management logic to maintain maximum overlap between channel activity and program instruction execution, that is, to keep the buffers full for as much of the time as possible. The buffers do not necessarily correspond to card images, but rather scatter read techniques employing the chain data facility of CCW's could be used to fill field specific buffers. In any case, the interface between the processor subroutine and the READ subroutine (Fig. 4-3) should consist of a user defined macro which when issued in the processor, calls the READ subroutine.

The address of a card image or the addresses of a group of fields from a card image would be returned to the processor in a parameter list. The buffer management routine would, of course, be responsible for synchronizing the EXCP I/O requests with the allocation of buffers.

An alternative approach to the relationship between processor and READ subroutine might be to have the processor specify the address or addresses into which the READ routine is to read the next card image or set of fields. The READ routine would then be required

to create dynamically, through instruction execution, the appropriate channel program, This arrangement implies that the buffer management function is to be included in the processor; otherwise the potential for overlap is greatly restricted.

In either approach the programmer's success in achieving CPU/channel overlap is indicated by the accumulated CPU wait time printed, together with the total execution time, at the end of each SALT job. The relationship between CPU wait time and I/O device characteristics can be explored by successive SALT runs in which the only change is the time parameters in the "DEVICE=" option on the $SALT card.

The lectures, while assignment #3 is in progress, might be concerned with the following topics:

(a) the structure of a supervisor as a collection of routines operating in real time in a logical relationship to each other determined by interrupts and by the execution of instructions for branching and loading of new program status words,

(b) a task (both system and program) represented by a collection of logically connected instructions related to other tasks through the interrupt mechanism,

(c) resources and their attributes,

(d) queues and the queuing of requests for a reusable resource,

(e) task supervision as an example of managing a resource, the CPU.

These topics form the background necessary for assignment #4

which consists of transforming the rudimentary interrupt routines and isolated software components of the previous assignment into a unified, embryonic, resident supervisor to service the needs of the problem program of assignment #3.

The structure of a minimal supervisor is shown in Fig. 4-4. Under this arrangement, an SVC primary interrupt handler determines the type of supervisor support requested and calls on the I/O supervisor in the event of an EXCP request or on the TASK supervisor for a WAIT request.

In essence, the I/O supervisor manages a queue of requests for each device (assuming one device on one channel) by enqueuing the CCB of each EXCP specifying the given device, and dequeuing a CCB on the occurrence of the I/O interrupt which indicates the end of the channel program associated with that CCB. Implied in the dequeuing procedure is the initiation of the channel program associated with the next CCB on the queue (unless the queue is empty) by execution of the privileged instructions SIO, TIO and HIO. Housekeeping functions such as the reflection of status information to the appropriate CCB after each I/O interrupt,and handling attempts to read past EOF or EOV belong in the I/O supervisor. The WRITE support routine can conveniently be included with the I/O supervisor although it could as readily be run outside the supervisor state entirely.

For this assignment the TASK supervisor is more rudimentary than the I/O supervisor. Its single function is to place the CPU into the WAIT state with I/O interrupts enabled whenever a WAIT

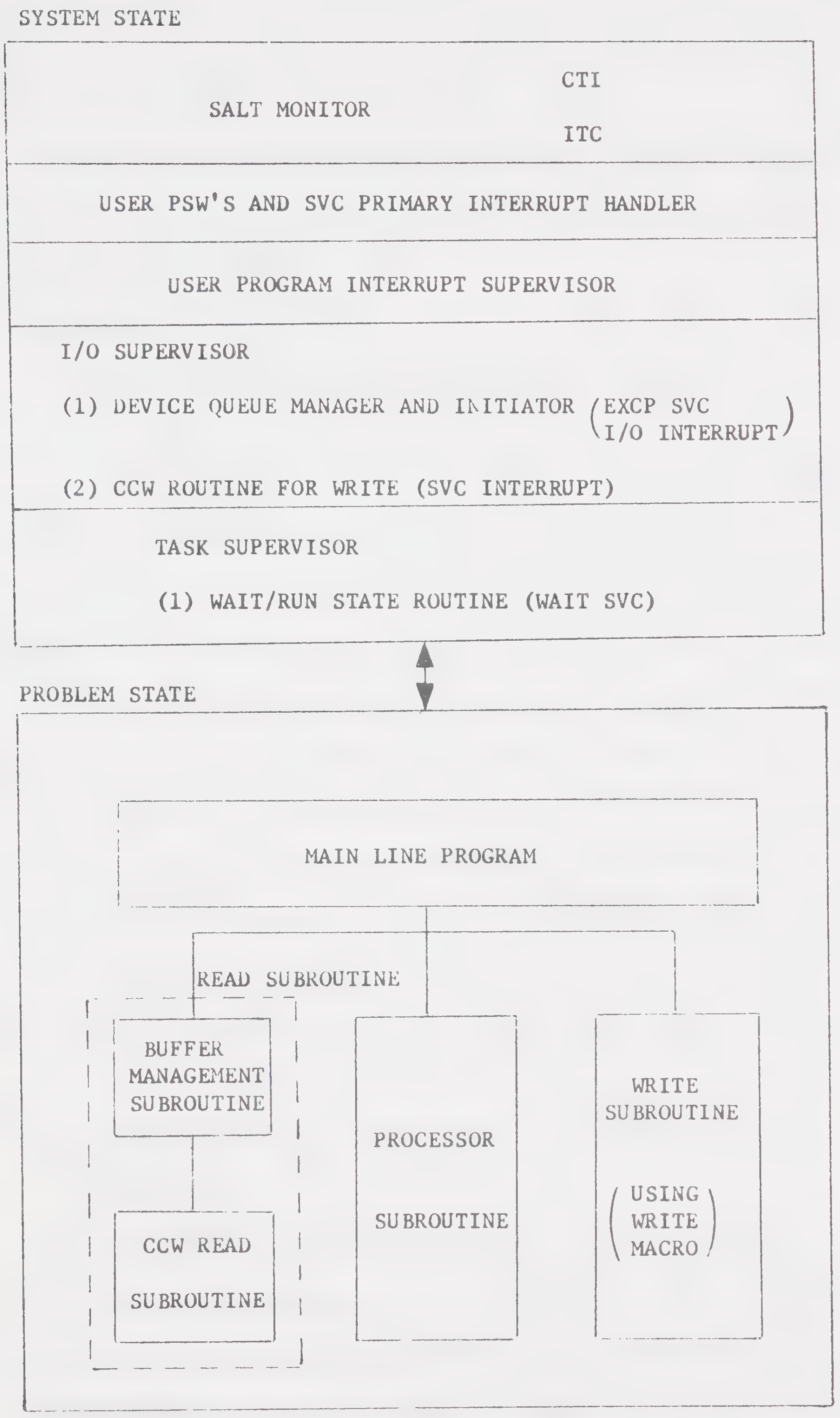

SYSTEM STATE
SALT MONITOR
CTI
ITC
USER PSW'S AND SVC PRIMARY INTERRUPT HANDLER
USER PROGRAM INTERRUPT SUPERVISOR
I/O SUPERVISOR
(1) DEVICE QUEUE MANAGER AND INITIATOR (EXCP SVC I/O INTERRUPT)
(2) CCW ROUTINE FOR WRITE (SVC INTERRUPT)
TASK SUPERVISOR
(1) WAIT/RUN STATE ROUTINE (WAIT SVC)
PROBLEM STATE
MAIN LINE PROGRAM
READ SUBROUTINE
BUFFER MANAGEMENT SUBROUTINE
CCW READ SUBROUTINE
PROCESSOR SUBROUTINE
WRITE SUBROUTINE
(USING WRITE MACRO)

Fig. 4-4. Fourth stage with user task and I/O supervisors

SVC has been executed.

The lectures while assignment #4 is being completed might center on non-supervisor system topics such as:

(a) assember design,

(b) compiler construction,

(c) loader and link-editor functions,

(d) utilities .

Assignment #5 (Fig. 4-5) is intended to illustrate this area. The problem consists of defining a very restricted assember language, a subset of the SALT language, and implementing the corresponding assembler as the processor subroutine of assignment #4. The language need not contain the facility for symbolic addressing, thus simplifying the assembly procedure to the following:

(a) scanning source statements obtained from the READ buffer manager to delimit the mnemonic and operand fields,

(b) checking fields for possible error conditions and issuing appropriate messages,

(c) converting mnemonic and operand fields to their SALT machine instruction equivalents,

(d) producing a simple line-at-a-time listing using the WRITE subroutine,

(e) loading each machine instruction as it is translated.

The lecture material, while the assembler is being developed, might deal with more advanced software characteristics such as:

(a) parallel, or asynchronous, processing of a number of tasks,

(b) spooling of input and output data,

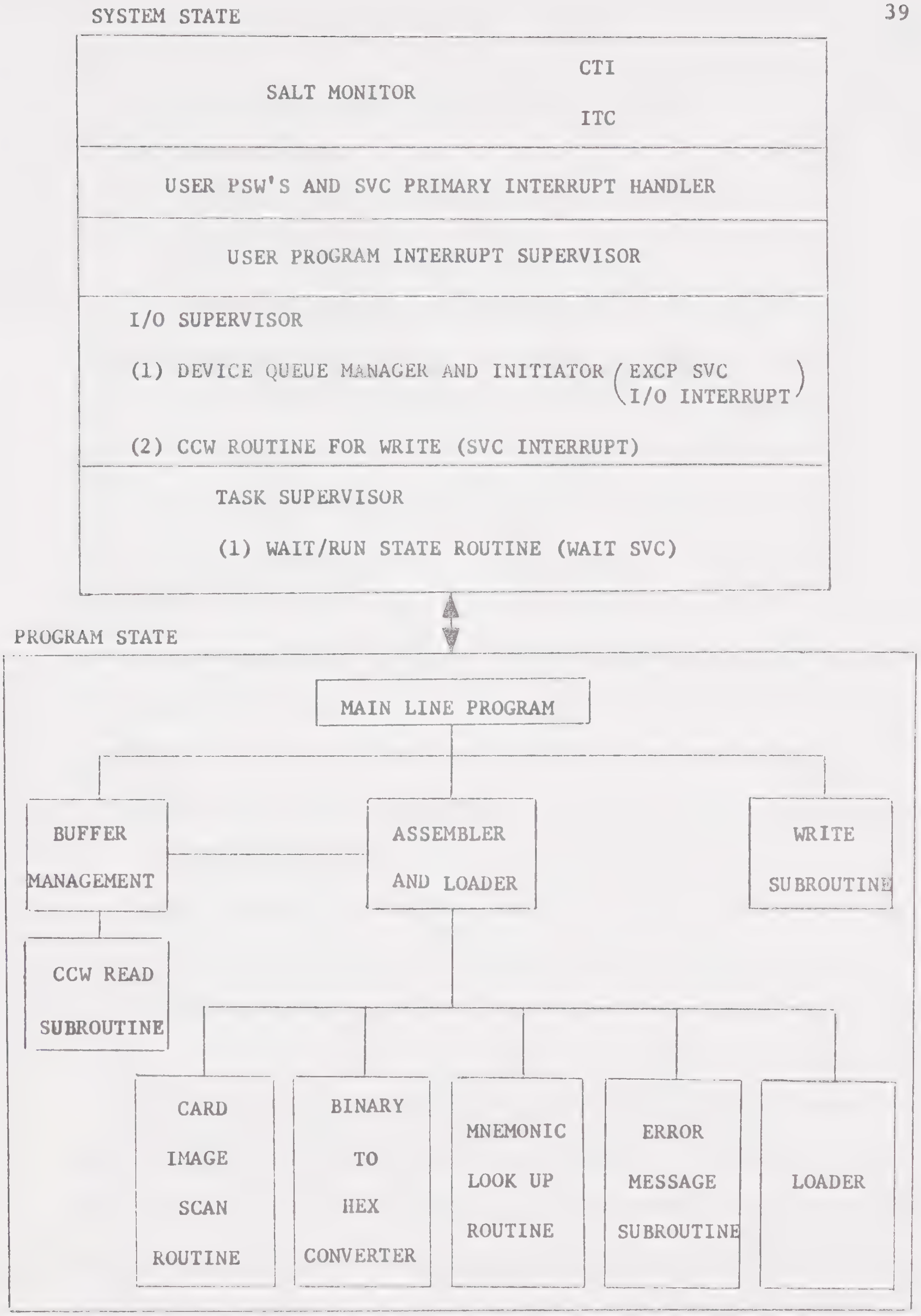


Fig. 4-5. Fifth stage employing user assembler for a greatly simplified assembler language

(c) spooling of jobs,

(d) time slicing,

(e) management of data structures,

(f) scheduling and management of jobs.

The final assignment in the program (#6, Fig. 4-6) entails a considerable degree of development from #5. A major change to the resident supervisor is required in order to expand the TASK supervisor from its rudimentary form in assignment #4 to include the capability of queuing Task Control Blocks of separately defined tasks to be run asynchronously, and to dispatch these tasks for CPU attention in an attempt to minimize CPU wait time.

An even more basic change is required to re-structure the problem program into a parent task that defines to the supervisor three subtasks:

(a) an asynchronous reader that spools card images into a large memory work area,

(b) an assembler/loader that uses the services of a buffer management routine to interface to the other tasks,

(c) an asynchronous writer that writes to the output device, lines placed in a large memory work area by the buffer management routine at the request of the assembler/loader.

The definition of the tasks to the supervisor, the relationship between tasks (Parent or Offspring) and the synchronization constraints between competing tasks should be accomplished by a set of user defined macros similar to the ATTACH, POST, WAIT, EXTRACT, DETACH group of system macros available under the IBM OS-MVT system.

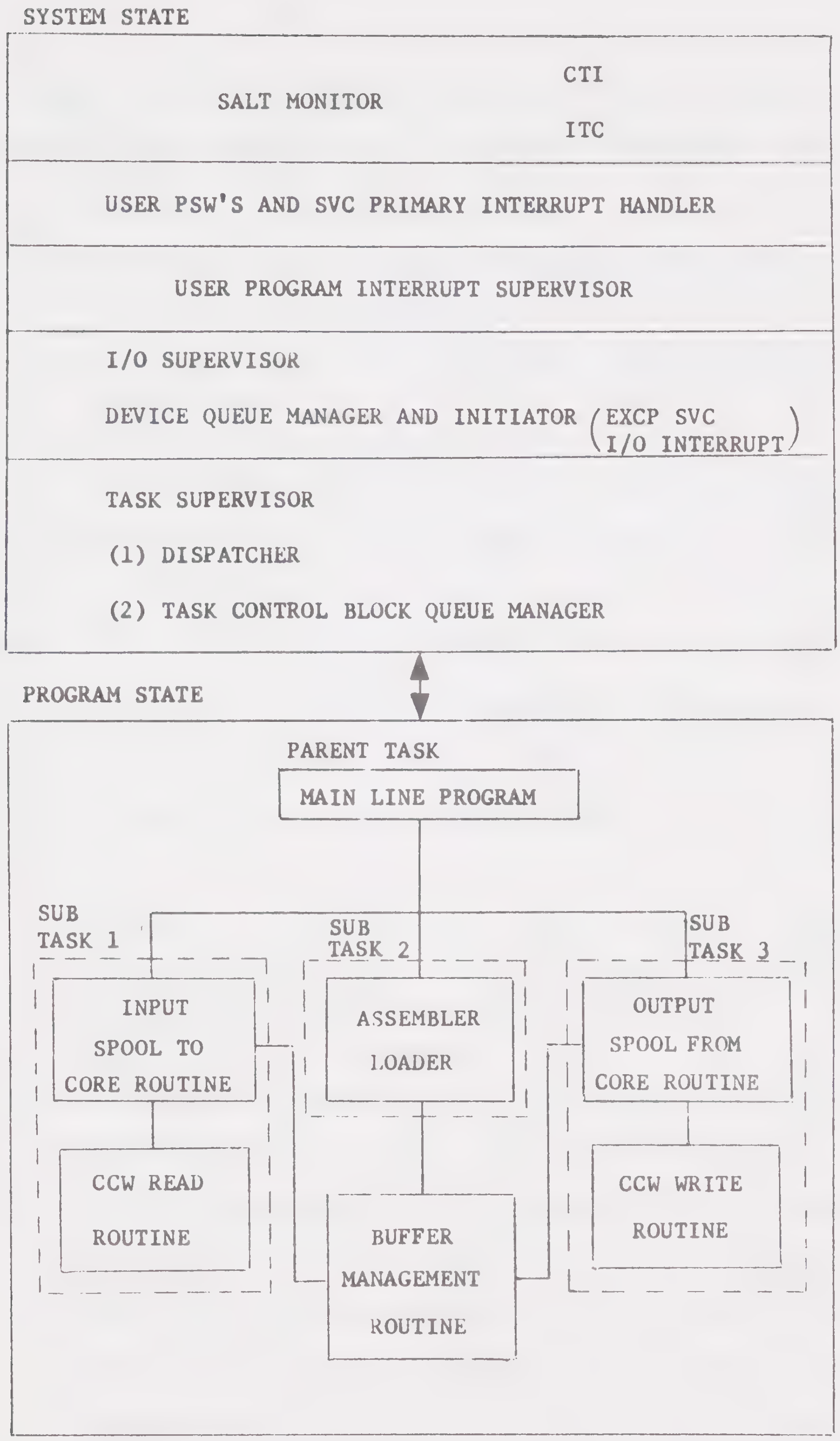

SYSTEM STATE
SALT MONITOR
CTI
ITC
USER PSW'S AND SVC PRIMARY INTERRUPT HANDLER
USER PROGRAM INTERRUPT SUPERVISOR
I/O SUPERVISOR
DEVICE QUEUE MANAGER AND INITIATOR (EXCP SVC I/O INTERRUPT)
TASK SUPERVISOR
(1) DISPATCHER
(2) TASK CONTROL BLOCK QUEUE MANAGER
PROGRAM STATE
PARENT TASK
MAIN LINE PROGRAM
SUB TASK 1
SUB TASK 2
SUB TASK 3
INPUT SPOOL TO CORE ROUTINE
ASSEMBLER LOADER
OUTPUT SPOOL FROM CORE ROUTINE
CCW READ ROUTINE
CCW WRITE ROUTINE
BUFFER MANAGEMENT ROUTINE

Fig. 4-6. Final stage containing user multiple task supervisor running device/memory spool packages asynchronously with assembler

After the combined system of assignment #6 becomes operational, a number of runs should be made to illustrate the dependency of system performance on the following factors:

(a) the size of the reader and writer memory work areas,

(b) the "DEVICE=" option parameters,

(c) the relative priorities between the reader, writer and assembler tasks.

The six preceding assignments represent a considerable amount of programming, probably too much to expect a student to complete in a one-term course. One solution would be to form groups in which the individuals develop different sections of the system. Another solution that appears to offer more advantages is to pre-program all six assignments and have each student create his own system, using selected routines from the pre-programmed version. The advantages of this are:

(a) The student is relieved of the work of writing and debugging parts of the system that are not interesting or particularly illustrative,

(b) Each individual can examine another programmer's work and thus become exposed to new techniques,

(c) The student is forced to abide by a standardized organization and module interface in order to use the pre-programmed routines, thus ensuring that he eventually develops a system that can be recognized as fulfilling the requirements of assignment #6.

4.4 Summary

The set of assignments outlined in the previous section is not intended to illustrate all possible uses of SUPERSALT, but the author feels they do demonstrate that the major problems associated with using the manufacturer's software can be overcome by using the SALT extension. The assignment descriptions and course outline are intended to describe an entirely feasible systems course plan and could be used as a guide in organizing future systems courses. As such, they reflect what the author believes to be important topics. For example, the hardware is certainly visible to the programmer of the six assignments, and it exists in a form sufficiently simple to enable the user to supply his own supervisor to accommodate his problem program. Hopefully, the user could create an increasingly sophisticated supervisor in response to the increasingly sophisticated requirements of his evolving problem program, thus gaining insight into both the application side and that of the supporting supervisor. At the same time, the lectures would no longer be a simplification, and therefore a flawed representation of a complex, pre-existing system, but rather would use the much simpler system developed in the assignments as a base from which to launch into a more complex and theoretical treatment of software organization.

CHAPTER V

IMPLEMENTATION OF SUPERSALT: THE MAJOR PROBLEMS

5.1 The Internal Facilities Required for CCW I/O Support

In order to properly simulate I/O operations to the user, the SALT system maintains tables containing descriptive and current status information for devices and channels defined, through the $SALT statement, to be part of the simulated machine. These tables are used to create and to maintain a stack of events, related to I/O activity, that provide the interface between the real input/output in real time and SALT input/output in SALT time.

For each possible device, there is defined a device description table whose format appears in Fig. 5-1. The first, third and fourth fields are inserted into the table from the fields of the 'DEVICE=' option (Appendix II) on the $SALT statement when it is scanned by the job control processor (JOBCNTRL) at the start of a SALT job. The second field contains a device type code used to determine the physical characteristics corresponding to this device. The device description tables are used by the system during execution of a SALT program to determine:

(a) if a unit on which I/O is requested actually exists,

(b) when, in SALT time, an initiated I/O operation should begin transmitting data across a channel,

(c) when, in SALT time, an initiated I/O operation should terminate.

The channel description table consists of a full word for each

of the eight possible channel designations (0 to 7). Each word contains a number indicating the "width" of the corresponding channel, that is, the number of bytes that can be transmitted across the channel in parallel. This information is used with the record length implied by the device type code in order to calculate the number of "pulse fronts" required by an I/O request that are to be serially transmitted across the channel.

For each device description table there is a device status table whose format is given in Fig. 5-2. The status tables reflect the instantaneous status of each I/O device within SUPERSALT at any given time. With the exception of the pointer to the CCB for this I/O operation, each field in the device status table is brought up to date after each pending event. That is:

(a) The second field is changed to point to the next (chronological) pending event for this I/O.

(b) Any change of status is reflected to the status field.

(c) The residual count field is decremented if information has been exchanged between a SALT program and an internal system buffer.

(d) The CCW address field may be changed if the current event specifies the end of a CCW which is chained to the next.

The channel status indicators consist of a double word with one byte reserved for each channel. A non-zero value implies that the corresponding channel is currently involved in an I/O operation.

The description and status tables are used in interpreting user-written CCW's. The problem can be adequately illustrated by

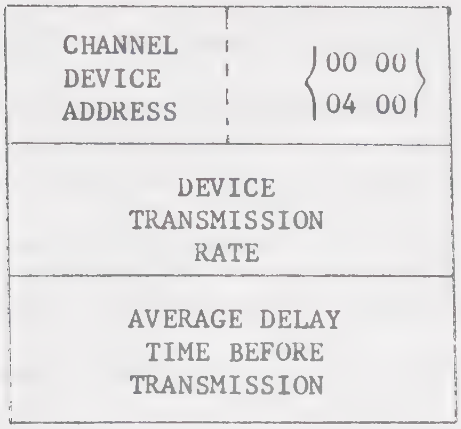


Fig. 5-1. The device description table

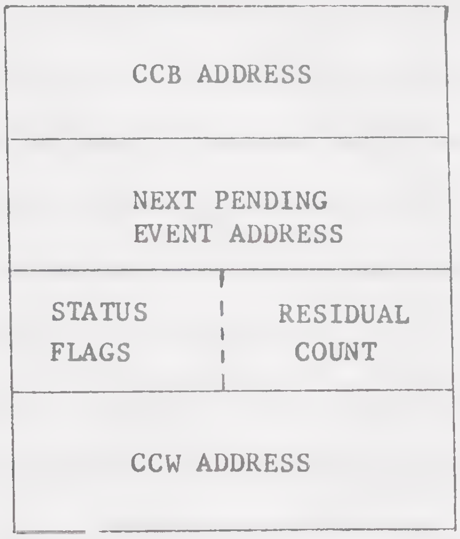


Fig. 5-2. The device status table

considering a user CCW employed to perform a simple card read. The read is initiated when the SVC associated with the EXCP macro is executed, causing a transfer of control into the SALT monitor. (If the user is handling his own interrupts, control passes to his I/O initiation routine and the I/O does not actually begin until the SIO instruction within that routine is executed.) After checking the validity of the command,device address, channel address and receiving area address, the monitor initiates an actual

read operation on a /360 channel. No further /360 instructions are executed within the SALT system until the read is finished.

The completion of the actual I/O is hidden from the SALT program, and SALT instruction execution is resumed. At a time synchronized to SALT instruction execution according to the device characteristics, "channel width" portions of information are "transmitted" from the internal system buffer to an address calculated from that specified in the CCW. It is the "remembering" of these "transmission" events that forms the most interesting aspect of the implementation. Each event consists of fields specifying:

(a) the time at which the event will occur,

(b) the type of event that will occur,

(c) the addresses between which data will be transferred,

(d) other control information,

and forms an element in a data structure. The type of data structure chosen was determined by the following requirements:

(a) The elements in the structure must be logically ordered in chronological sequence.

(b) Since previously initiated I/O may be in progress when a given I/O operation is initiated,the data structure must allow the interleaving of new elements "between" old elements to preserve the ordering.

(c) As events "happen" they should be easily removed from the structure and the memory space they occupied made available for re-use.

(d) Since channel execution of a CCW can be halted before nor-

mal termination (HIO), it must be possible to remove efficiently from the structure all events associated with a given channel program even though they may not be logically adjacent, that is, events for other I/O may intervene.

The above requirements clearly dictate that the data structure be a form of linked list with a main linkage that links the events in chronological sequence and a secondary linkage that links events for the same CCW from first to last. Since events may be of several types requiring different amounts of information, the list elements must be variable in size.

The memory space for the data structure is dynamically allocated from the high end of the region at the beginning of execution of each SALT job, and thus reduces the space available for the machine instructions of SALT programs. It is therefore necessary to make the most efficient use of the space allocated to the data structure. A traditional method [12] of managing space allocation for a linked list is, in effect, to keep two lists; the second linking the free space interspersed within the allocated elements. As new elements enter, the list space for them is obtained from the free memory list and as elements are deleted, the space they occupied may be immediately returned to the free memory list or left for collection by the later execution of a garbage collector.

Because of the extremely active nature of the data structure when a SALT program with large I/O requirements is run, the garbage collection method was eliminated from consideration. It was felt

the collection routine would be called to rebuild the free storage list too frequently in the restricted memory space, resulting in high overhead. The other method of attaching elements to the free storage list as they were deleted would ordinarily be suitable, but some problems are encountered in the case of a list with variable length records in that list elements of differing sizes are required from, and released to, the free list. In order to keep memory fragmentation to a minimum, the free list would need to be maintained in order of size of element from smallest to largest, and some scheme should be implemented to consolidate two or more adjacent elements into one free list element. For a highly active structure, this procedure entails a considerable amount of searching lists and relinking of elements into and out of them.

An alternative method to these traditional solutions was employed to minimize overhead. This method involves restricting the variable sized elements to multiples of eight bytes and employing an allocation control block in which every bit indicates by its value the current status of a particular eight byte field in the data structure (the pending event stack). That is, if the third bit is one, then the third double word in the data structure is currently being used. The procedure for allocating space for an element requiring, for example, three double words is as follows:

(a) Starting at the left of the allocation control block, scan toward the right, looking for the first three consecutive bits of value zero.

(b) When these are found (assuming they are), set them to value

one and calculate the corresponding address in the pending event stack as follows:

> address of free element = (number of bits offset into the allocation control block × 8) + address of pending event stack

(c) In the newly allocated element save the one byte mask field used to "OR" the allocation bits to one to facilitate the eventual freeing of the space; the element can then be created and logically linked into the stack.

The calculation of the free element address can be performed very efficiently in a /360 since the value of the offset ends up in a register. The multiplication by eight is a left shift of three positions, and the addition of the result with the stack address can be achieved with a simple "load address" instruction. However, designing a procedure for efficient scanning of the allocation control block for consecutive zero bits presented a considerable challenge.

The procedure for de-allocating the space occupied by a newly deleted element consists of:

(a) calculating the number of bytes offset into the allocation control block required in order to use the one byte mask stored during allocation.

> number of bytes offset = (address of this event - address of start of stack)/64

Division of 64 can be achieved by a right shift of six positions.

(b) "exclusive ORing" the mask byte with the appropriate allocation control block byte to set the bits corresponding to the memory reserved for the element to zero, freeing the area for re-allocation.

The layout of a typical pending event is shown in Fig. 5-3. This event controls transfer of information between the system buffer and user area. Other event types have a similar format and all have identical fields in the first sixteen bytes. Other event types and their uses are:

(a) Control - to cause, at the device, some form of mechanical activity other than reading or writing records, for example, spacing the printer,

(b) I/O Interrupt - to cause a SUPERSALT I/O interrupt to mark the end of a channel program,

(c) Chain Command Marker - to mark the last event of a CCW that had the chain command flag on, and to contain the address of the CCW "chained" to,

(d) Chain Data Marker - to mark the last event of a CCW that contained a one bit in the chain data flag and to point at both the next CCW to be processed and at the position in the device buffer to which, or from which, the I/O operation is to continue.

0	2	3	4	8	9	12	13	14	16	20
LINK	NO. OF BYTES	TYPE	TIME	NXT EVENT	CCB ADDR	COM	ALLOC MASK	STATUS TAB DISP	FROM ADDR	TO ADDR

LINK: contains the displacement to the next chronological event

NUMBER OF BYTES: specifies the number of bytes of information to be transmitted by this event

TYPE: indicates by its value that this event is to cause a transfer of information

TIME: specifies the SALT time at which this event is to occur

NXT EVENT: contains the double word displacement to the next event in the sequence for this CCW

CCB ADDR: points to the CCB associated with this event

COM: contains the command code for this CCW

ALLOC MASK: contains the bit pattern required to free the memory occupied by this event after the event has "occurred"

STATUS TAB DISP: contains the displacement to the status table for this event

FROM ADDR: specifies the address from which "NO. OF BYTES" of information is to be transmitted

TO ADDR: specifies the address to which "NO. OF BYTES" is to be transmitted

Fig. 5-3. The fields of a pending event to control information transfer

The correspondence between a typical CCW and the events that represent it are shown in Fig. 5-4. As a result of the execution of the SVC for the EXCP, the SALT system calls on the CCW processor which inspects the CCW and determines that:

(a) Two information transfer events of twenty bytes each would exhaust the count field of the CCW.

(b) The first event should be placed in SALT time "Average Delay Time" microseconds beyond the current SALT time.

(c) The other events should follow at (channel width × constant)/(data transfer rate) intervals.

(d) The last event should indicate that chain data was requested, and point at the CCW chained to.

Following creation and insertion of these events into the Pending Event Stack, the SALT system continues with SALT instruction execution. After execution of each instruction, an appropriate value is added to the accumulated SALT time. Between instructions, the Pending Event Stack is checked to see if the next event is scheduled to "happen" now. In the example, the first event would be "accomplished" by moving twenty bytes from the system buffer to LINE, while the third event would involve moving twenty bytes to LINE + twenty from the system buffer address + twenty. The fourth event would involve the fetching of a new CCW and the calling of the channel processor to process it. The new CCW contains the address into which the READ is to continue while the current position in the system buffer would be recorded in event four.

From the above example it is seen that only one CCW from a chan-

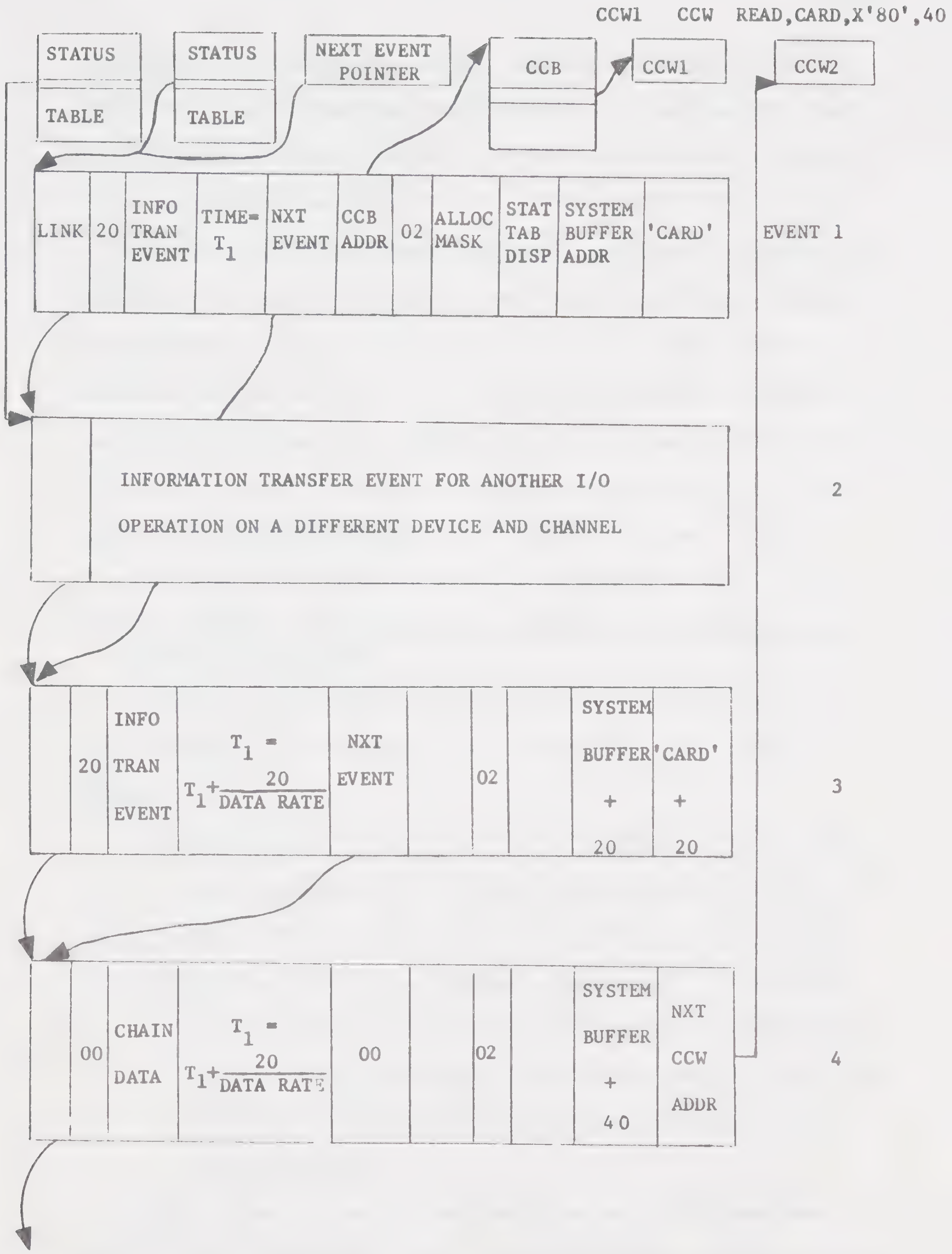


Fig. 5-4. Pending events corresponding to a CCW initiated at time T_1 minus "Average Delay Time"

nel program is processed at a time. This is necessary, both to reduce the number of pending events maintained at any one time and because the execution of one CCW in a chain could be used to read in the next CCW in the chain.

In practice, the preceding scheme has worked well for a machine configuration including one eighty byte record input device and one 120 byte record output device. With twenty byte channel widths, the worst case situation is ten pending data transfer events plus two pending I/O interrupt or chain events which require a total of 272 bytes. In fact, a double word is reserved for allocation bit indicators, implying a present stack size of 512 bytes.

5.2 The Internal Facilities Required to Simulate a Multi-State CPU with Interrupt System

Since SALT instructions are interpretively executed rather than executed on /360 hardware, the simulation of a CPU can be achieved by:

(a) monitoring the state of the CPU as presented in the current status indicators to determine whether instruction execution should proceed, and, if so, what instruction set is valid,

(b) creating a SUPERSALT interrupt whenever an SVC instruction is executed, or between instructions whenever an I/O interrupt type of pending event is serviced or whenever an invalid condition is detected during interpretive execution of a SALT instruction.

If the "PSW=" option has been specified on a $SALT statement, then the SALT interpreter, prior to SALT instruction execution, moves

the first eight bytes from the address given in the operand field of the option to an internal field used as the current program status word (PSW). This process is somewhat similar to the /360 initial program load procedure. If the initial PSW specifies "run" state, then SALT instruction execution proceeds from the address specified in the PSW until the first interrupt. An interrupt is simulated by:

(a) moving the current PSW field to the Old PSW slot,

(b) moving the New PSW slot to the current PSW field,

(c) moving the SUPERSALT general purpose register fields to the Old GPRS slot,

(d) moving the New GPRS slot to the SUPERSALT register fields,

(e) proceeding with instruction execution (if run state has been entered) under control of the new current PSW.

The WAIT state is simulated by ceasing instruction interpretation until the next SUPERSALT interrupt can be forced to occur by advancing the SALT time scale. This is done through clearing any pending events from the stack, one after another, advancing SALT time appropriately in each instance until an I/O interrupt on an unmasked channel is encountered. If none is found before the stack is exhausted, the program is terminated for running overtime in a locked WAIT condition.

The process of masking a channel and thus postponing any I/O interrupts from that channel is simulated by leaving the appropriate I/O interrupt event in the stack unserviced past its scheduled time. Between each instruction an attempt will be made to

service this interrupt. The attempt will be successful if the mask has just been removed; otherwise the next event in the stack is checked to see if it is scheduled to occur now. A masked-off I/O interrupt can therefore remain at the head of the stack while events that become current are serviced from "behind" it.

Implementation of routines to interpret privileged SALT instructions presented few difficulties and they are only briefly described here.

The routine for SIO simply calls the CCW processor and provides a dummy internal CCB. It uses the return information from the CCW processor to set the condition code and status table entries, and may move status information from the device status table to the user-supplied Channel Status Word.

The routine for HIO extracts the address of the first pending event (if any) for the specified device, releases the memory for it and re-links its neighbours, uses the NXT EVENT field to calculate the next pending event in the stack for this device, deletes it and continues following the chain until the last pending event for this device has been removed from the stack. The routine then resets the status table, thus releasing the device for subsequent I/O operation.

The routine for TIO instruction extracts status information from the device status table and inserts it into the channel status word.

The routine for the SSM instruction simply resets the system mask field of the field containing the current SUPERSALT PSW with a

specified bit pattern.

The routine for the LPSW instruction moves the sixteen words from the New GPRS field provided by the user to the SUPERSALT register fields in the interpreter and moves eight bytes from the address specified to the current PSW field.

5.3 Variations in the System Format

The extensions to the system were implemented in a form that uses the conditional assembly feature of the /360 Assembler Language. By appropriate settings for two symbolic parameters and subsequent re-assembly, the following versions of the SALT system can be generated from the same source program:

(a) The fully extended system: this version has support for channel programming and user interrupt handling. If both the "DEVICE=" and "PSW=" options are employed in a job, then that job is restricted to a maximum about 10K smaller in object size than under the original SALT system, and that job may require up to twice (depending on I/O demands) the /360 CPU time for SALT program interpretation. If only the "DEVICE=" option is specified, then the CPU overhead is slightly reduced. If only the "PSW=" option ss specified, implying that only READ's or WRITE's are used in the program, then CPU overhead is only slightly degraded from the original SALT system.

(b) The semi-extended system: this version has support for channel programming. The "PSW=" option is not recognized

as valid on the $SALT card and no internal current PSW is maintained in the system nor is an interrupt mechanism simulated. The CPU overhead is slightly less than using (a) with the "DEVICE=" option only, while memory size restrictions are only slightly improved over (a).

(c) The basic system: this system has no support for user interrupt handling or channel programming. The "DEVICE=" and "PSW=" options are both invalid on the $SALT card. The user sees the system as described in Easton and Penny [10], and internally it is little changed from the system described in Dutton [7]. The CPU overhead and memory requirements are exactly the same as those of the original SALT system.

In accordance with good software design principles, the three systems are compatible in one direction. That is, a program designed for and run on system (c) can be run unchanged on systems (a) and (b). Similarly, a program run on system (b) can be run unchanged on system (a).

CHAPTER VI

LIMITATIONS OF SUPERSALT AND PROPOSALS FOR ITS FUTURE DEVELOPMENT

6.1 Limitations

A software system can be evaluated by examining the extent to which it fulfills its intended roles. The original aim in creating SUPERSALT was to produce an operational, distributable system such that:

(a) The user would be convinced that he is programming a complete computer, that is, one in which there are no inconsistencies in the time base.

(b) The user would be aware of the characteristics of, and responsible for the activity of, peripheral devices and channels operating asynchronously with the CPU.

(c) The user would have the facility for developing within the simulated machine, his own software systems on virtually any level of complexity from basic interrupt handlers for selected types of interrupts to full-scale operating systems.

(d) The user would be able to explore the relationships between various machine configurations and the software characteristics necessary to utilize the hardware effectively.

(e) The above characteristics would be supported in a system that protects the installation from the user and makes modest demands for memory space, /360 CPU attention and /360 I/O resources.

The following discussion points out that these aims have not been entirely fulfilled, and Section 6.2 describes extensions to SUPERSALT designed to overcome these shortcomings. The proposals in the next section were not implemented in the original extension in order to allow time to evaluate, through experience gained in using SUPERSALT, the precise form that future developments should take. However, for each development proposed, a general plan of implementation is provided to illustrate its feasibility.

The simulated computer system, specified in Chapter III, appears to meet the requirements stated in (a) and (b). Close examination reveals, however, an inconsistency in time that is apparent only when one considers the complete SALT job consisting of assembly and execution. Essentially, the problem is that the SALT source program is processed by an assembler running on a /360 machine while the resulting SALT object program is executed immediately afterward on a simulated, simplified /360 (SUPERSALT) which executes instructions in the order of 100 times slower than the actual /360 Model 65. In other words, a SALT job is completely assembled and loaded into core, and only then is the initial PSW inserted into SUPERSALT to cause its activation. The single SALT object program might contain a supervisor for the SUPERSALT CPU and several programs recognized by that supervisor. The termination of the run is caused by execution of an SVC 0 while in the supervisor state, which appears to deactivate the SALT computer and, in fact, causes the SALT system to perform job-to-job transition.

One possible way of viewing this situation such that the incon-

sistencies become more acceptable is that there are really two computers, a SUPERSALT CPU and a /360 CPU, both sharing the same memory. The assembler is assumed to run on the /360, assembling a SALT job. After loading it into memory, the /360 gives a "tap on the shoulder" to the SUPERSALT CPU, causing the initial PSW to be loaded and the SALT program executed. The execution of an SVC 0 on the SUPERSALT CPU while in the supervisor state is assumed to cause a "shoulder tap' back to the /360 which has been in the WAIT state, and then assembly of the next job proceeds. Similarly, other supervisor functions reflected to the SALT monitor can be considered as receiving service from the /360 CPU.

The aims expressed in (d) and (e) above are only partially realized in SUPERSALT. It is certainly true that the user can create his own interrupt handler to explore the operation of the multi-state CPU as is done in Appendix V. However, it is certainly not true of the extension described here that the user can create his own full operating system to virtually any level of complexity. Those functions of an operating system that can be demonstrated by taking an instantaneous view of the operation of the system can, in general, be accommodated by SUPERSALT. A typical example of this type of function is the creation of asynchronous tasks within a job. It is quite feasible to create a supervisor that supports, and a problem program that requests, subtasking, and have the supervisor and program assembled and executed as the same SALT job. In general, only those functions that can be illustrated as operating in an interrupted fashion over a comparatively long time scale are not

adequately supported by SUPERSALT. Examples of this type are scheduling and initiating jobs and passing information from job to job through data sets.

For example, it is quite possible as outlined in Chapter IV, to create a software system that contains a supervisor which runs an assembler and an input stream of pseudo programs for assembly and execution, all within one SALT job. This arrangement implies that modularity can only be achieved through separate tasks connected by PSW exchanges or through subroutines of the same assembly. It is not feasible to:

(a) cause the output of one execution of the assembler to be combined with a subsequent output to form a single object program,

(b) create a data set in one program that is to be used by a subsequent program,

(c) maintain libraries of data sets global to all programs within a SALT batch,

(d) maintain libraries of data sets global to all SALT batches,

(e) cause the input to or output from a SALT job to be spooled between devices,

(f) have student designed data set organization schemes used within SALT jobs.

Clearly, if some form of pseudo device were supported which allowed read/write data sets whose records were independently or nearly independently accessible, the limitations mentioned above could be largely overcome. Further, if this device were interfaced to /360

direct access devices, the shortcoming mentioned in (d) above could be eliminated.

The final aim, (e), mentioned at the beginning of the chapter has been met. The installation is fully protected from the user, SUPERSALT will run in 100K of memory or even less for small SALT programs, /360 CPU requirements (even though perhaps doubled from the original system for some jobs) are still many times less than would be the case using the manufacturer's software for the same purpose, and /360 I/O requirements have not increased over the original SALT system. For example, the sample program in Appendix V was run in a region of 100K and was assessed a total charge of $.25 by the charging system in use at the University of Alberta under the OS/MVT system.

6.2 Recommendations for Further Development

It is apparent from the discussion in the previous section that the largest potential improvement in the system could be obtained by providing some form of simulated direct access device that is programmable by the user. The internal organization of the system as set out in Chapter IV does not preclude the addition of such a device, providing the characteristics of the I/O are carefully defined with a view to simplicity. One such definition follows.

Records, the unit of information, are individually addressable on such a device through specifying a compound address, the parts of which can be considered as specifying the nodes of a tree structure representing the physical organization of the device (Fig. 6-1).

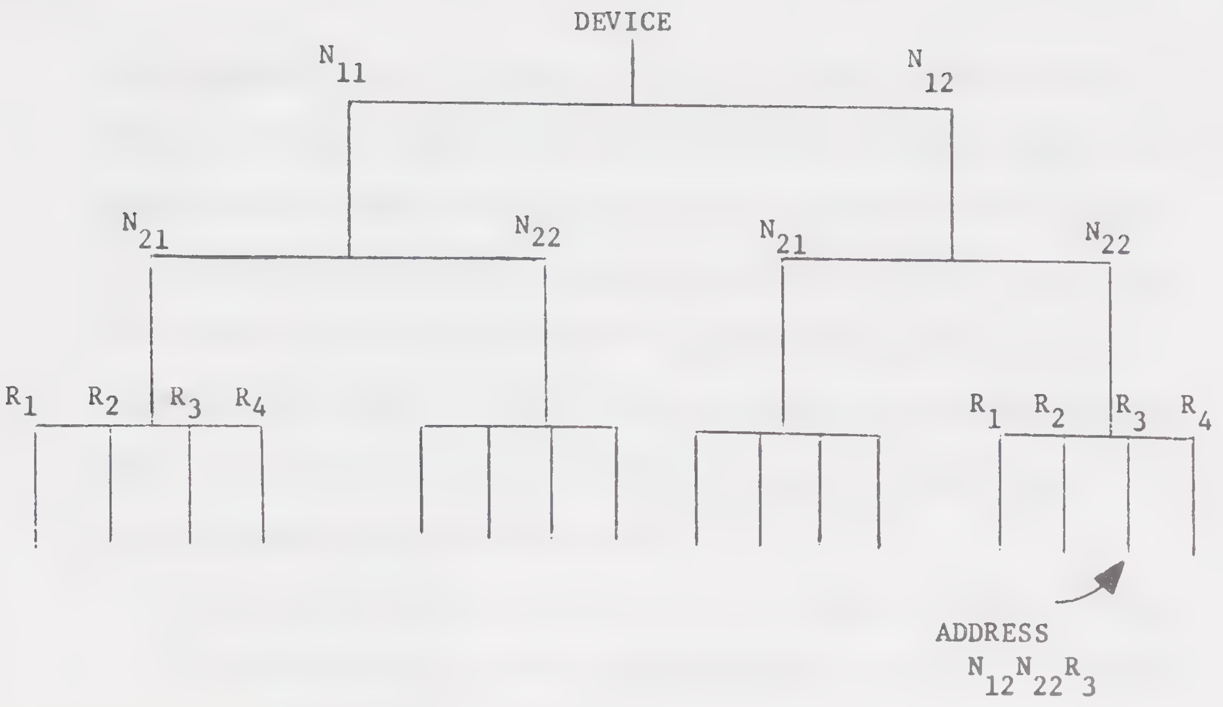


Fig. 6-1. A direct access device addressing scheme

The device is assumed to have physical mechanisms that require the channel, through execution of CCW's, to cause explicitly the shift from one node to another, either down or across. For example, if the last record read, using the notation of Fig. 6-1, was $N_{11}N_{21}R_4$, the record $N_{12}N_{22}R_3$ could be read by the following channel program using command chaining:

(a) a CCW to place the mechanism at node N_{12},

(b) a CCW to place the mechanism at node N_{22},

(c) a CCW to place the mechanism at record R_3,

(d) a CCW to cause the record "pointed at" to be read.

If a second READ were issued without repositioning the mechanism, the next record, R_4, would be obtained. A third READ would retrieve record $N_{11}N_{21}R_1$. If a READ or WRITE were issued in which the count

field exceeded the record size, only that record would be read or written, and the status would indicate that the record length was exhausted before the count field, with the number of bytes remaining in the count specified by the residual count field. In brief, this device is a unit record device in which the records are re-readable, re-writable and individually addressable. Ideally, the device is assigned through the "DEVICE=" option, describing not only the device address and type but:

(a) the number of bytes in a record and the data transfer rate;

(b) the number of records grouped under a N_{2-} node and the time required to position at a record after positioning at the N_{2-} node,

(c) the number of N_{2-} nodes grouped under a N_{1-} node and the time required to position at an N_{2-} node after positioning at the appropriate N_{1-} node,

(d) the number of N_{1-} nodes and the time to proceed from one to another.

The implementation of such a device seems to be entirely feasible using the current internal organization specified in Chapter V. The device characteristics specified on the $SALT card could be kept in an extended device description table and used by the CCW processor in managing the Pending Event Stack. The status table would require an extension to keep information on the current position of the read/write mechanism. The pseudo device could be interfaced to the real installation direct access devices through the SALT system maintaining an actual data set whose records each contain,

say, a complete record group under a N_{2-} node and are individually accessible. The procedure for simulating the set of CCW's outlined above for reading record $N_{12}N_{23}R_3$ after reading $N_{11}N_{21}R_4$ would be:

(a) Process the first CCW and enter two pending events into the stack, one to cause a change of address in the status table from $N_{11}N_{21}R_4$ to $N_{12}N_{21}R_1$ and one to cause command chaining, and flag the status table indicating a disk read will be required.

(b) When the chain command event is serviced, process the second CCW and enter two pending events into the stack, one to change the mechanism from address $N_{12}N_{21}R_1$ to $N_{12}N_{23}R_1$ and the other to cause command chaining, and ensure the status table is flagged for a disk read.

(c) When the chain command event is serviced, process the third CCW and enter two events into the stack, one to change the mechanism from address $N_{12}N_{23}R_1$ to $N_{12}N_{23}R_3$ and one to cause command chaining.

(d) When the chain command event is serviced, process the fourth CCW and since it specifies READ and the READ flag is on in that status table, read into a system buffer the actual disk record corresponding to $N_{12}N_{23}R_-$, calculate the address of R_3 and move it to another system buffer, and insert the necessary events into the stack to move the required number of bytes of the buffer to the SALT program followed by an I/O interrupt event to mark the end of the channel program.

Even though the implementation of a device like the above seems

entirely feasible in the context of the extended system structure, several hazards are apparent from an operational point of view. They are:

(a) If a user specifies an unreasonably large record size or an unreasonably large number of records within each N_2-node, then a large SALT system buffer will have to be reserved, and for the case of large record size an unreasonable number of pending events will have to be accommodated on each I/O operation.

(b) A user may, through carelessness or error, cause more /360 disk operations to be performed than are necessary, thus tying up system resources,

(c) A user may, through appropriate /360 JCL, cause disk data sets created during a SALT program to be kept almost indefinitely.

Clearly, internal limits on record size and numbers must be set to minimize overhead, and adequate control measures over disk space and usage maintained to avoid contradicting aim (e) stated at the beginning of the chapter.

A second addition to the system which would perhaps not provide quite as much improvement in the flexibility of the system as the direct access support just discussed, would be a provision to run parts of a SALT program on different SUPERSALT CPU's sharing a common memory and performing in a multi-processor configuration. To the user it would appear that he was programming two or more CPU's, each of whose reserved core area was specified on the SALT statement

in a separate "PSW=" option field, all of which may have access to any of the channels or devices described in the "DEVICE=" options and all of which appear to run asynchronously with, and semi-independently from, each other. Fully independent operation, of course, would defeat the purpose of the addition since the illustrative aspect of developing software for multiprocessing systems is controlling the communication and "sometimes" synchronization between all of the CPU's through their interrupt system in such a way as to use the total resource efficiently. Asynchronous operation would naturally not be the case whenever there was memory contention between CPU's, that is, no CPU could fetch (either for instruction execution or during instruction execution) or store a byte of memory that was simultaneously being fetched or stored by another CPU. The implementation of a "Test and Set" instruction like the /360 instruction of the same name would seem advisable to allow control of a common resource by the multiple CPU's.

The implementation of this second addition again seems entirely feasible and depends upon making the SALT interpreter re-entrant. Separate current PSW's, register fields and instruction address registers would have to be kept for each SUPERSALT CPU and the interpreter would have to be "called" to interpret one instruction from each CPU instruction stream in turn. If the current PSW of a particular CPU specified the WAIT state, then no instruction would be interpreted in that CPU in contrast to the other still functioning CPU's, providing the impression to the external world of asynchronous operation. Only one device description and device status block per

device specified on the $SALT statement would be maintained, since all CPU's share the same physical devices. One undesirable characteristic does appear if, to maintain realism, all the CPU's are considered to operate in the same SALT time, that is, if there is only one pending event stack containing the events for all CPU's, and these events are made to happen along a time base tied to instruction execution time in all the CPU's. This implies that for "n" CPU's, each having an instruction interpreted in turn, the single SALT time base must be incremented by the execution time of the longest instruction of the "n" executed. The hardware can be considered as having a common clock pulse generator that causes all the CPU's to perform instruction fetching simultaneously. Any CPU that finishes executing an instruction before the others must await the next common instruction fetching cycle.

This unusual characteristic, although not very realistic, does not appreciably detract from the advantages offered by the opportunity to develop software systems to control a multiprocessor configuration. However, there is a strong probability that aim (e) stated at the beginning of the chapter would, with such a system, be only partially obtained. Almost certainly the SALT programs would become so much larger that added demands would be made on /360 memory space, and the additional overhead necessitated by a re-entrant interpreter coupled with the huge number of instructions that might be interpreted for multiple CPU's would cause a dramatic increase in the amount of /360 CPU attention required.

A third addition to the system might involve the addition of sup-

port in the SALT monitor for asynchronous subtasks within a SALT job. This would require the inclusion of system macros with the following functions:

(a) Initiate Task (ITASK) - indicates to the SALT monitor that the address of the task control block specified in the operand contains information necessary to define an asynchronous subtask as an offspring of the originating parent task,

(b) Task Control Block (TCB) - a table of information relevant to a subtask. The information includes the entry point address, the latest register contents whenever the task is waiting, and fields used by the WAIT and POST macros,

(c) Wait (WAIT) - allows a subtask or originating task to wait on the posting of a completion of an event from the subject task as indicated in its TCB,

(d) Post Completion of an Event (POST) - allows a subtask to communicate the completion of an event to all tasks waiting on it,

(e) Post Completion of a Subtask (RETRN) - provides a means of removing a subtask from active competition for CPU attention.

The above three additions compose what the author feels would be the most valuable extensions from the standpoint of instruction potential. It should be stressed, however, that the first two would have to be implemented carefully and used prudently to avoid wasting the /360 resources.

CHAPTER VII

CONCLUSION

While the study of systems is beginning to be recognized as an important part of Computer Science curricula [2], little attention has been given to the problems of illustrating lecture material through assignments. As was described in Chapter II, the typical student-oriented low level language processor lacks the realistic time base and simulated CPU characteristics to act as the vehicle through which assignments could be undertaken. The author's experience in using the manufacturer's software as the basis on which to set assignments merely demonstrates how efficiently the manufacturer's hardware/software combination isolates the user, preventing him from directly viewing the very aspects with which lectures on systems are concerned. The most direct solution, that of presenting each student with real hardware for which he must develop software, suffers from (in addition to economic problems) a surfeit of unenlightening "real life" complexities and details which would make assignments even more difficult and time-consuming without adding to their worth as a teaching aid.

The alternate solution, that of extending a student processor in order to present to the student the appearance of a simplified computer system, has been investigated from the point of view of:

(a) the essential characteristics of such a system for purposes of illustration,

(b) its usefulness as a mechanism around which assignments for

a first course in computer system could be designed,

(c) its shortcomings and omissions (and their possible solutions), particularly as support for advanced systems courses,

(d) the problems encountered in the implementation of a working system.

The preceding chapters have largely ignored the feasibility of such a system from an economics point of view. No really detailed analysis can be made comparing the cost of using the manufacturer's software with that of using SUPERSALT, for running assignments, until the extended SALT system has been used in this capacity. However, the author has noted that the use of OS-MVT software for assignments in a previously presented systems course cost between \$1.00 and \$10.00 per run, depending on the assignment. They also required the use of temporary disk files. The far more extensive SUPERSALT assignments outlined in Chapter IV would, in the author's opinion, cost between \$.10 and \$2.00 per run and would require I/O only on system input and output devices. While it is not the function of this thesis to establish the limits of economic feasibility for any course, it may be concluded that SUPERSALT offers a more economic and more versatile mechanism for running assignments for a systems course than has hitherto been employed.

The author further concludes that this investigation has established both the feasibility of creating such a system and the desirability of using it as the foundation on which a course in computer systems could be structured.

BIBLIOGRAPHY

1. Achtemichuk, L. (1967), "Survey of Computer Science Courses at Canadian Institutes", Supplement to the Quarterly Bulletin, vol. 7, 4.

2. ACM Curriculum Committee on Computer Science, (1968), "Curriculum 1968 (Recommendations for Academic Programs in Computer Science)", Communications of the ACM, vol. 11, 3, pp. 151-197.

3. BCS Working Party 7, (1967), "Survey of Computer Science Courses", Computer Bulletin, vol. 11, 1, pp. 31-42.

4. Center for Computer and Information Sciences, (1968), SOS: The Brown University Student Operating System, Brown University, Providence, Rhode Island.

5. Dutton, J.B. (1971), "Programming the SUPERSALT Computer", Computing Center Publication, University of Alberta, (in preparation).

6. Dutton, J.B. (1970), "A Pseudo Clock for SALT", Computing Review, University of Alberta Student Chapter of the Association for Computing Machinery, vol. 3, pp. 33-46.

7. Dutton, J.B. (1969),"Student Assembler Language Translator System Description", Computing Center Publication, University of Alberta.

8. Dutton, J.B. (1971), "SUPERSALT System Description", Computing Center Publication, University of Alberta (in preparation).

9. Dutton, J.B. and Penny, J.P. (1970), "Undergraduate Education in the Fundamental Concepts of Large Computer Systems", Proceedings IFIP World Conference on Computer Education, Amsterdam, vol. 2, pp. 185-189.

10. Easton, L.S. and Penny, J.P. (1969), "Student Assembler Language Reference Manual", Department of Computing Science, University of Alberta.

11. Jones, P.D. (1967), "Operating System Structures", Proceedings IFIP Congress, Edinburgh, pp. 29-38.

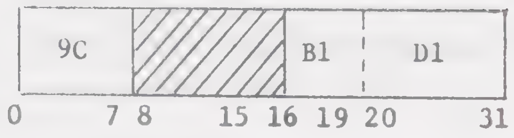
12. Schorr, H. and Waite, W.M. (1967), "An Efficient Machine-Independent Procedure for Garbage Collection in Various List Structures", Communications of the ACM, vol. 10,8, pp. 501-506.

13. Wile, D.S., Munchk, R.G. and Van Dam, A. (1967), "The Brown University Student Operating System", Proceedings of the ACM National Meeting, pp. 427-439.

APPENDIX I

ADDITIONS TO THE MACHINE INSTRUCTION SET

1. Start I/O - (SIO) -



SI

The execution of this instruction signals the addressed channel (if available) to fetch eight bytes from the address given in the Channel Address Word and to use these eight bytes as a Channel Command Word directed to the addressed device.

The D1, B1 fields of the instruction are not used to generate a memory address. Instead, the low order sixteen bits of the sum formed by the addition of the contents of register B1 and the contents of the D1 field identify the channel which is to fetch a CCW and the device to which the CCW is to apply.

The I/O operation is initiated if:

(a) the addressed channel exists and is not currently active performing a previously initiated I/O operation, and

(b) the addressed device exists and is not currently active performing a previously initiated I/O operation.

A channel and device are considered busy until the interrupt marking the end of the operation has occurred. Thus, if that interrupt is masked off, the channel and device are unavailable even though the I/O operation may have been completed at the device.

The resulting condition code indicates the status of the I/O attempt.

Condition Code:

0 - I/O operation initiated successfully and channel now in operation

1 - the addressed channel found to be busy controlling I/O on the addressed device

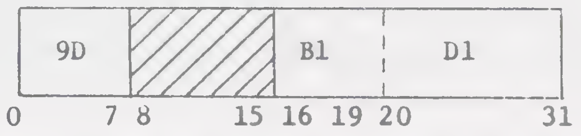
2 - the addressed channel found to be busy performing I/O on some device other than the addressed device

3 - the I/O could not be initiated - a Channel Status Word indicating the reason is stored

Program Interrupt:

Privileged Operation

2. <u>Test I/O - (TIO)</u> -



SI

The execution of this instruction gives indication of the state of the addressed channel and device by causing the condition code to be set and the instantaneous Channel Status Word to be stored.

The D1(B1) fields of the instruction are not used to generate a memory address. Instead, the low order sixteen bits of the sum formed by the addition of the contents of register B1 and the contents of the D1 field identify the channel and device that are to be tested.

Condition Code:

0 - both the addressed channel and device presently free

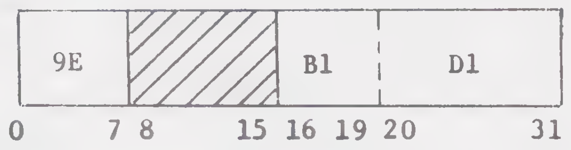
1 - the addressed channel found to be busy controlling I/O on the addressed device

2 - the addressed channel found to be busy performing I/O on some device other than the addressed device

3 - no test could be made but a Channel Status Word is stored anyway, indicating the reason

Program Interrupt:

Privileged Operation

3. <u>Halt I/O - (HIO)</u> - 

SI

The execution of this instruction causes the immediate termination of any I/O operation in progress on the addressed channel and device, and causes the instantaneous Channel Status Word to be stored. The results of the instruction execution are indicated by the setting of the condition code.

The D1(B1) fields of the instruction are not used to generate a memory address. Instead, the low order sixteen bits of the sum formed by the addition of the contents of register B1 and the contents of the D1 field identify the channel and device to which the instruction applies.

Condition Code:

0 - the addressed device found to be not engaged in I/O activity

1 - the I/O is complete, but the device and channel are not free since a masked I/O interrupt has been stacked in hardware before execution of the HIO - the interrupt can only be cleared by setting the system mask in the current PSW to unmask the addressed channel

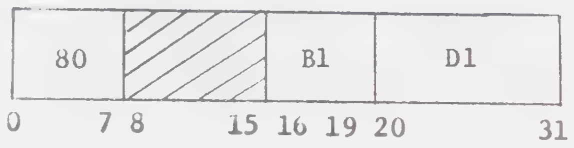
2 - the I/O operation in progress on the addressed device has been halted and that device and the addressed channel are now free

3 - an invalid condition exists and is specified in detail in the stored Channel Status Word

Program Interrupt:

Privileged Operation

4. <u>Set System Mask - (SSM)</u> -



The byte at the memory location designated by the operand address replaces the system mask of the current PSW.

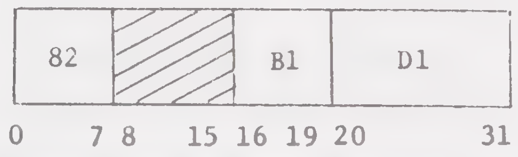
Condition Code: Unchanged

Program Interrupts:

Privileged Operation

Addressing

5. <u>Load Program Status Word - (LPSW)</u> -



The eight bytes at the full-word aligned location specified by the operand address completely replaces the current PSW which is lost. The sixteen general purpose registers are loaded from a memory location reserved for this purpose (see Appendix II). Instruction execution continues under control of the newly loaded PSW at the location contained in the instruction address field of the new PSW. This adadress is not checked for validity during execution of the LPSW but is checked during execution of the following instruction. The LPSW

is executed under control of the old current PSW which must have the CPU in the supervisor state. The newly loaded PSW, however, can place the CPU into the problem state without contradicting the privileged nature of the LPSW instruction.

Condition Code:

The condition code is set according to bits 34 and 35 of the newly loaded PSW.

Program Interrupts:

Privileged Operation

Addressing

Specification

Note: In the event of a program interrupt during execution of an LPSW instruction, the PSW stored in the Program Old PSW slot is the old current PSW.

APPENDIX II

SUPERSALT HARDWARE CHARACTERISTICS

1. Specification of I/O Device Characteristics

Hardware specification for the devices and channels attached to the SUPERSALT CPU are detailed on the $SALT card in special option fields of the form:

$$\text{DEVICE=}\left\{\begin{array}{l}\text{READER}\\ \text{PRINTER}\end{array}\right\}\text{,addr,datarate,avedelay}$$

where

READER - indicates an input unit record device of fixed record length of 80 bytes

PRINTER - indicates an output unit record device of fixed record length of 120 bytes plus one byte carriage control

addr - a three character hexadecimal constant of the form abb where "bb" is the device address, "a" is a number 0 to 7 and is the address of the channel to which the device is connected

datarate - the rate of transfer of information through the device and down the channel in bytes per second

avedelay - the average delay time in microseconds before the command is carried out at the device

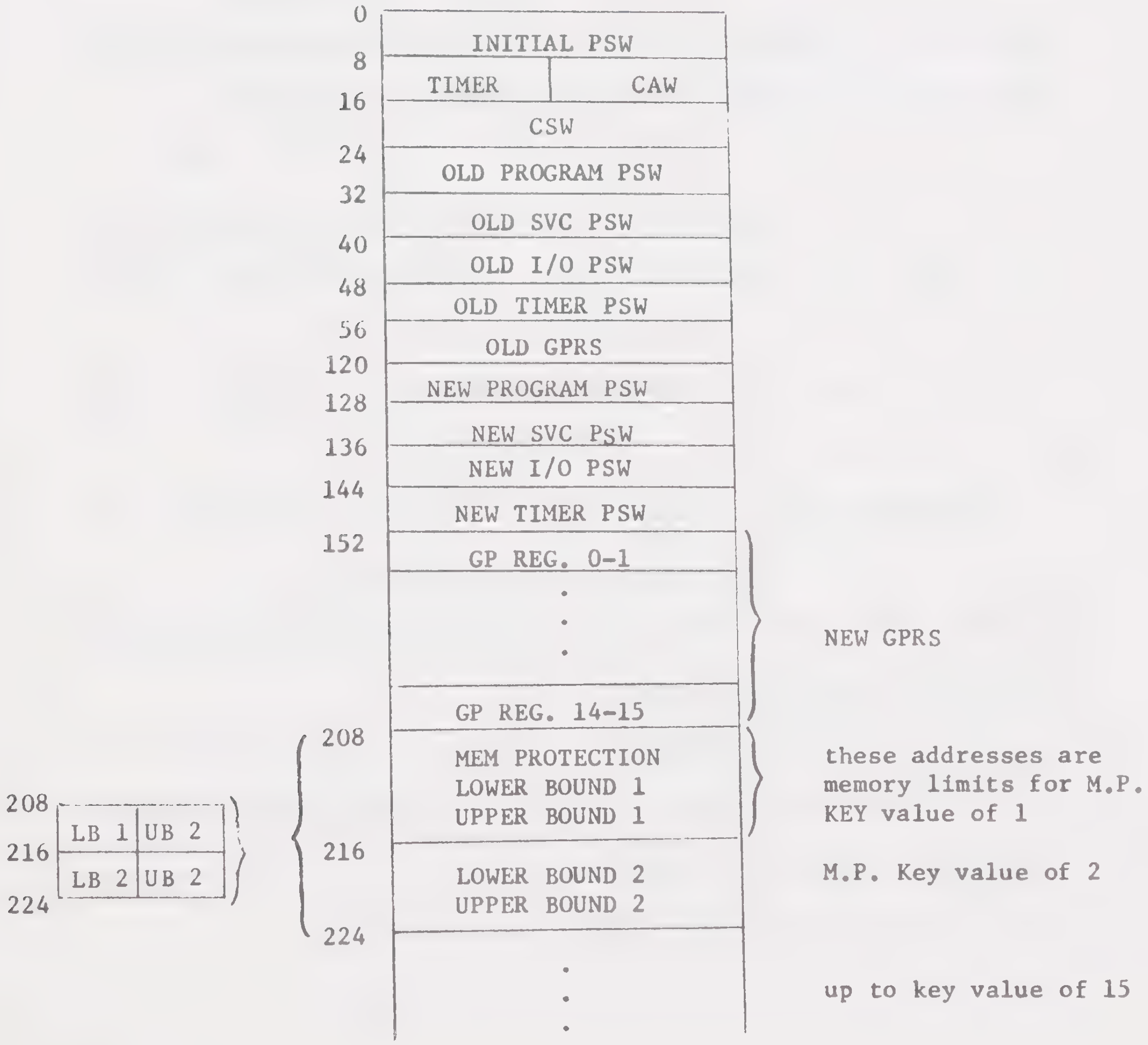
2. Specification of CPU Reserved Memory Format

The user may provide his own PSW's by including on the $SALT card the following option field:

PSW=addr

where

addr - is the relative address of a memory area in the user's SALT program which is considered to have the following format:



where

INITIAL PSW - is the PSW loaded at the beginning of execution of the SALT program.

TIMER - is a full word that may be set to any value by the programmer. It is decremented in synchronous with execution of SALT instructions and causes a timer interrupt whenever the value goes from zero to a negative value or when it goes from the maximum negative number to zero (not currently implemented).

CAW - (Channel Address Word) set by the programmer to the address of a Channel Command Word before an SIO instruction is executed.

CSW - Channel Status Word; see No. 3 following.

OLD 'X' PSW's - the location where the current PSW is stored during an 'X' class interrupt.

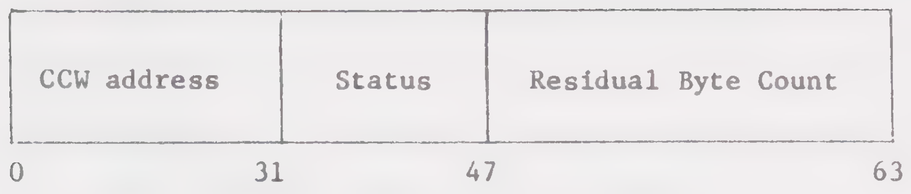
NEW 'X' PSW's - the location where a new PSW is loaded from for an 'X' class interrupt. See No. 4 below for PSW format.

G.P. REG 0-15 - contents of user's general purpose registers; stored by hardware during an interrupt.

MEM PROTECTION LOWER BOUND 'X' - the address below which memory is store-protected when the PSW Memory Protect Key value is 'X'.

MEM PROTECTION UPPER BOUND 'X' - the address above which memory is store-protected when the PSW Memory Protect Key value is 'X'.

3. <u>Channel Status Word Format</u>



where

CCW Address - is the address of the Channel Command Word that was being executed or had just completed execution at the time the CSW was stored.

Status - indicates the status of the channel at the time the CSW was stored. Bits 32 to 39 indicate status information resulting from normal channel operation. Bits 39-47 indicate abnormal channel status. A one bit indicates the following:

<u>bit</u>

32 - unused

33 - unused

34 - the current (or last) CCW specifies (ed) chain data

35 - the current (or last) CCW specifies (ed) chain command

36 - end of file has been detected

37 - the count value originally specified in the count field of the current CCW has been exhausted before the end of the physical block.

38 - the end of the physical block has been detected at the device before the count value, originally specified in the count field of the current CCW, has been exhausted.

39 - the status information just stored is the instantaneous status of a continuing I/O operation

40 - unused

41 - unused

42 - the addressed channel does not exist

43 - the addressed device is not one of those controlled by the addressed channel

44 - command not recognized by the addressed device

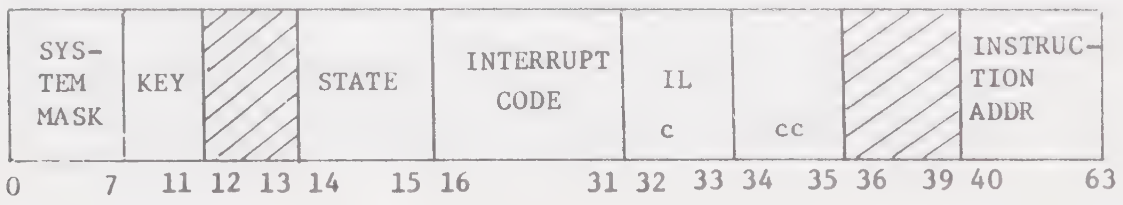
45 - 'control' command contains invalid modifier bit combination

46 - command not recognized by the addressed channel

47 - attempt to read past physical end of device

Residual Byte Count - the difference between the count value obtained from the count field of the current CCW and the number of bytes transferred down the channel (up to the time the CSW was stored) under control of that CCW's command.

4. <u>Program Status Word Format</u>



A one bit in any of the following positions indicates:

<u>bit(s)</u>

0 - any I/O interrupt associated with an I/O operation on this

channel is postponed until a new PSW in which this bit is zero is loaded

1 - 7 same for channels 1 to 7

14 - the CPU is in the WAIT state (RUN state when the bit is zero)

15 - the CPU is in the PROBLEM state (SUPERVISOR state when the bit is zero)

16 - 31 the code for the interrupt that caused the storing of this PSW, that is; for an SVC interrupt the SVC code, for an I/O interrupt the channel/device address, and for a program interrupt the program interrupt code

32 - 33 instruction length code for the current instruction

00 - 2 bytes	RR
01 - 4 bytes	RX
10 - 4 bytes	RS-SI
11 - 6 bytes	SS

34 - 35 condition code

40 - 63 address of current instruction

APPENDIX III

THE CHANNEL COMMAND WORD

1. <u>The Assembler Instruction</u>

```
name    CCW    command,addr,flag,count
```

where

COMMAND is an absolute expression specifying the command to be executed by the channel. The channel command set consists of

0000 0001 - write

0000 0010 - read

0000 1000 - transfer in channel

0000 MM11 - control printer

MM = 00 skip page

01 space 3

10 space 2

11 space 1

ADDR is a relocatable expression specifying the address where:

(a) the channel is to transmit data to (READ),

(b) the channel is to retrieve data from (WRITE),

(c) the channel is to retrieve the next CCW from (TRANSFER IN CHANNEL).

FLAG is an absolute expression that specifies the flags for bits 32-36. and zeros for bits 37-39 of the machine command (see below).

COUNT is an absolute expression that specifies the number of bytes of

data to be acted upon by this command.

2. <u>The Machine Command Format</u>

The SALT assembler creates from the CCW statement an eight-byte field, aligned on a full-word boundary, having the following format:

COMMAND CODE	DATA ADDRESS	FLAGS	00	COUNT
0 7	8 31	32 39	40 47	48 63

where a one bit in the following position indicates:

<u>bit</u>

32 - Chain Data: this command is to apply (without reference to the CPU) to the COUNT and DATA ADDRESS fields of the next contiguous CCW, following completion of the current CCW.

33 - Chain Command: the channel is to fetch and execute (without reference to the CPU) the next contiguous CCW, following completion of the current CCW. (However, if the current CCW has caused EOF on an input device, an I/O interrupt terminates automatic channel fetching even though command chaining is specified.)

34 - Unused

35 - Skip: for this CCW, data transmission across the channel is to be suppressed; however, the command is to be carried out at the device.

36-39 - 0 (unused)

NOTE: Only command chaining can be carried across a TRANSFER IN CHANNEL.

APPENDIX IV

ADDITIONS TO THE SYSTEM MACRO SET

1. Execute Channel Program

```
name        EXCP        ccbname
+           L           1,=A(ccbname)
+           SVC         6
```

Description: The EXCP macro requests the SALT monitor (or a user-supplied supervisor if the "PSW=" option was specified on the $SALT card) to initiate an I/O operation on the issuing program's behalf. The Channel Control Block whose address is placed in register 1 is assumed to contain the

(a) address of the channel to be used,

(b) address of the required device on that channel, and

(c) the address of the CCW to be used by the channel.

2. Test and Wait on I/O

```
name        WAIT        ccbname
+           L           1,=A(ccbname)
+           TM          8(1),X'01'
+           BZ          *+6
+           SVC         7
```

Description: The WAIT macro is used to impose whatever synchronizing constraints are required between program instruction execution

and channel operation. The Channel Control Block, whose address is placed in register 1, is assumed to contain a status bit set to one by the monitor (or supervisor) whenever the CCB was named in an EXCP for an I/O operation that is currently in progress and set to zero following the I/O interrupt terminating that I/O operation. When the WAIT macro is issued for a CCB flagged as in use, the WAIT SVC is executed, providing the monitor (supervisor) with the opportunity to wait on completion of the I/O.

3. Channel Command Block

```
 name      CCB      X'deviceadr',ccwname
+          DC       A(ccwname)
+          DC       X'deviceadr'
+          DC       2X'00'              RESIDUAL COUNT
+          DC       2X'00'              STATUS
+          DC       2X'00'              SPARE
```

where

(a) deviceadr is four hexadecimal digits of the form ccdd

cc - channel address 00 to 07

dd - device address 00 to FF

(b) ccwname is the symbolic address of the CCW associated with this CCB.

Description: The CCB provides a means of communicating shared information between the issuing program and the resident monitor or supervisor. A given CCB is specific to a given unit (deviceadr) and a

given CCW chain (ccwname); however, more than one CCB can refer to the same unit or CCW chain. The residual count field is used by the monitor or supervisor, on completion of the I/O operation referencing this CCB as a field in which to store the difference between the count value obtained from the count field of the last CCW in the chain and the number of bytes transmitted down the channel during execution of that CCW. The status field is used by the monitor or supervisor to reflect to the issuing program the status of the last completed I/O operation referencing this CCB, if the unit is presently free, or the current I/O operation, if it is busy. A one bit in the following status field positions indicate:

Bit	
0 - 1	(unused)
2 -	the current CCW specifies chain data
3 -	the current CCW specifies chain command
4 -	the last completed I/O operation caused end of file
5 -	the count value obtained from the count field of the final CCW in the chain of CCW's last completed was exhausted before the end of the physical block
6 -	the last I/O operation caused the end of the physical block at the device before the count value, obtained from the final CCW in the chain of CCW's last completed, was exhausted
7 -	this CCB is being used by an I/O operation currently in progress.

APPENDIX V

A SAMPLE SALT PROGRAM

The following computer listing is that of a SALT program run on the SUPERSALT machine. The program contains an elementary supervisor consisting of simple interrupt handlers to service the needs of a problem program employing a variety of different channel programs. The hardware configuration was specified on the $SALT card with fields specifying:

```
DEVICE=READER,1C1,120000,100

DEVICE=PRINTER,2C2,220000,200

PSW=4
```

```
                                                   MINI SUPERVISOR-RESERVED CORE FORMAT                                  PAGE    1
  LOC     OBJECT CODE      ADDR1  ADDR2    STMT    SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

000000                                        2           START
000000                                        3           USING *,3
                                              4   *-----------------------------------------------------------------------
                                              5   ***   RESERVED CORE SETTINGS
000000  00000000                              6   SUPER    DC    F'0'            SPARE
                                              7   ***   INITIAL PSW
000004  FF110000                              8            DC    X'FF110000'     PROB/RUN STATES-INT'S ENABLED-PROT KEY 1
000008  00000530                              9            DC    A(USENTRY)      USER ENTRY ADDR
                                             10   ***   TIMER
00000C  00989680                             11   TIMER    DC    F'10000000'     NOT IMPLEMENTED
                                             12   ***   CHANNEL ADDRESS WORD
000010  00000000                             13   CAW      DC    F'0'
                                             14   ***   CHANNEL STATUS WORD
000014  0000000000000000                     15   CSW      DC    2F'0'
                                             16   ***   OLD PROGRAM PSW
00001C  0000000000000000                     17   OPPSW    DC    2F'0'
                                             18   ***   OLD SVC PSW
000024  0000000000000000                     19   OSVCPSW  DC    2F'0'
                                             20   ***   OLD I/O PSW
00002C  0000000000000000                     21   OIOPSW   DC    2F'0'
                                             22   ***   OLD TIMER PSW
000034  0000000000000000                     23   OTPSW    DC    2F'0'
                                             24   ***   OLD GENERAL PURPOSE REGISTERS
00003C  0000000000000000                     25   OGPRS    DC    16F'0'
                                             26   ***   NEW PROGRAM PSW
00007C  00000000                             27            DC    X'00000000'     SUP/RUN STATES-INT'S MASKED- PROT KEY 0
000080  000000F4                             28            DC    A(PROGPROC)     PROGRAM INTERRUPT HANDLER ADDR
                                             29   ***   NEW SVC PSW
000084  00000000                             30            DC    X'00000000'
000088  000002A4                             31            DC    A(SVCPROC)      SVC INTERRUPT HANDLER ADDR
                                             32   ***   NEW I/O PSW
00008C  00000000                             33   NIOPSW   DC    X'00000000'
000090  000004CE                             34            DC    A(IOPROC)       I/O INTERRUPT HANDLER ADDR
                                             35   ***   NEW TIMER PSW
000094  00000000                             36            DC    X'00000000'
000098  000000E4                             37            DC    A(TIMPROC)      TIMER INTERRUPT HANDLER ADDR
                                             38   ***   NEW GENERAL PURPOSE REGISTERS
00009C  000000000000000000                   39   NGPRS    DC    3F'0'           GPR 0-2
0000A8  00000000                             40            DC    A(SUPER)        GPR 3     BASE REGISTER FOR SUPERVISOR
0000AC  0000000000000000                     41            DC    12F'0'          GPR 4-15
                                             42   ***   MEMORY PROTECTION LIMITS
0000DC  00000530                             43            DC    A(USENTRY)      KEY=1  STATIC ALLOCATION-
0000E0  00000918                             44            DC    A(USENTRY+1000)    -OF 1000 BYTES
                                             45   ***   END OF RESERVED CORE
                                             46   *-----------------------------------------------------------------------
```

```
                                                    MINI SUPERVISOR-INTERRUPT HANDLING ROUTINES                    PAGE   2

  LOC     OBJECT CODE       ADDR1  ADDR2    STMT    SOURCE STATEMENT             VERSION 2 U OF A  SALT   4 DECEMBER 1970

                                               48   *------------------------------------------------------------------------
                                               49   ***   TIMER INTERRUPT HANDLING ROUTINE
000F4   D203  300C  30F0  0000C  000F0         50   TIMPROC  MVC   TIMER(4),HIVALU    BOOST TIMER
0000FA  8200  3034               00034         51            LPSW  OTPSW              CONTINUE WITH PREVIOUS PSW
0000F0  3B9ACA00                               52   HIVALU   DC    F'1000000000'
                                               53   *------------------------------------------------------------------------
                                               54   ***   PROGRAM INTERRUPT HANDLING ROUTINE
0000F4  9E00  02C2               002C2         55   PROGPROC HIO   X'02C2'
                                               56            DUMP  SUPER,256      DUMP OLD PSW'S AND GPRS
0000F8  4110  3000               00000         57 +          LA    1,SUPER
0000FC  4100  0100               00100         58 +          LA    0,256
000100  0AC5                                   59 +          SVC   5
000102  1BAA                                   60            SR    R10,R10
000104  43A0  301F               0001F         61            IC    R10,OPPSW+3   GET INT CDE
000108  89A0  0002               00002         62            SLL   R10,2         TIMES 4
00010C  58AA  31CC               001CC         63            L     R10,MADTAB(R10)    MESSAGE ADDR
000110  D213  3161  A000  00161  00000         64   CANJOB   MVC   MSGSLOT(20),0(R10)
000116  9E00  02C2               002C2         65            HIO   X'02C2'        SCRAP PREVIOUS IO
00011A  9E00  01C1               001C1         66            HIO   X'01C1'
00011E  41A0  3144               00144         67            LA    R10,PINTCCW
000122  50A0  3010               00010         68            ST    R10,CAW
000126  9C00  02C2               002C2         69            SIO   X'02C2'
00012A  4770  314C               00140         70            BC    7,OUT+8
00012E  D203  3090  313C  00090  0013C         71            MVC   NIOPSW+4(4),OUT+4
000134  8200  3138               00138         72            LPSW  OUT
000138                                         73            DS    0F
000138  FF020000                               74   OUT      DC    X'FFC20000'    SUP/WAIT STATES-INT'S ENABLED-PROT KEY 0
00013C  C0000140                               75            DC    A(*+4)         EOJ ADDR
                                               76            EOJ
000140  0AC0                                   77 +          SVC   0
000144  CF0001544000000                        78   PINTCCW  CCW   SPACE1,ERLINE,CCOM,0
00014C  0100015400000078                       79            CCW   WRITE,ERLINE,NOFLAGS,120
000154  C1C2C5D5C440C4                         80   ERLINE   DC    C'ABEND DUE TO '
000161  40404040404040                         81   MSGSLOT  DC    107C' '
                                               82   *
0001CC  00000000                               83   MADTAB   DC    A(0)
0001D0  000001F0                               84            DC    A(DIVMSG)      INT CODE 1
0001D4  00000204                               85            DC    A(PRIVMSG)              2
0001D8  00000218                               86            DC    A(OPERMSG)              3
0001DC  0000022C                               87            DC    A(OVFLMSG)              4
0001E0  00000240                               88            DC    A(SPECMSG)              5
0001E4  00000254                               89            DC    A(REGMSG)               6
0001E8  00000268                               90            DC    A(OPRNDMSG)             7
0001EC  0000027C                               91            DC    A(PROTMSG)              8
                                               92   *
0001F0  C4C9E5C9C4C540                         93   DIVMSG   DC    C'DIVIDE EXCEPTION    '
000204  D7D9C9E5C9D3C5                         94   PRIVMSG  DC    C'PRIVILEGED OPERATION'
000218  C9D5E5C1D3C9C4                         95   OPERMSG  DC    C'INVALID OP CODE     '
00022C  D6E5C5D9C6D3D6                         96   OVFLMSG  DC    C'OVERFLOW            '
000240  C1C4C4D9C5E2F2                         97   SPECMSG  DC    C'ADDRESS ALIGNMENT   '
000254  D9C5C740E207C5                         98   REGMSG   DC    C'REG SPECIFICATION   '
000268  D6D7C5D9C1D5C4                         99   OPRNDMSG DC    C'OPERAND ALIGNMENT   '
00027C  D7D9D6E3C5C3E3                        100   PROTMSG  DC    C'PROTECT VIOLATION   '
000290  C9D5E5C1D3C9C4                        101   IOMSG    DC    C'INVALID IO CONDITION'
```

```
                                             MINI SUPERVISOR-INTERRUPT HANDLING ROUTINES                        PAGE   3
  LOC     OBJECT CODE      ADDR1  ADDR2    STMT    SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

                                            102 *------------------------------------------------------------------------
                                            103 ***   SUPERVISOR CALL (SVC) INTERRUPT HANDLING ROUTINE
0002A4  1BAA                                104 SVCPROC  SR    R10,R10
0002A6  43A0  3027                00027     105          IC    R10,OSVCPSW+3       GET SVC NO.
0002AA  89A0  0002                00002     106          SLL   R10,2               TIMES 4
0002AE  59A0  32DC                002DC     107          C     R10,SVCTBLE   TEST VALIDITY
0002B2  4720  3482                00482     108          BH    BADSVC
0002B6  58FA  328C                002BC     109          L     R15,SVCTBL(R10)
0002BA  07FF                                110          BR    R15
0002BC  000002E0                            111 SVCTBL   DC    A(SVCEOJ)      SVC 0
0002C0  000002F2                            112          DC    A(SVCREAD)         1
0002C4  00000320                            113          DC    A(SVCWRITE)        2
0002C8  00000398                            114          DC    A(SVCCTI)          3
0002CC  000003AA                            115          DC    A(SVCITC)          4
0002D0  000003BC                            116          DC    A(SVCDUMP)         5
0002D4  000003CE                            117          DC    A(SVCEXCP)         6
0002D8  00000482                            118          DC    A(SVCWAIT)         7
0002DC  0000001C                            119 SVCTBLE  DC    A(SVCTBLE-SVCTBL-4)
                                            120 *------------------------------------------------------------------------
                                            121 *        END OF JOB-REFLECT SVC TO SALT MONITOR
                                            122 SVCEOJ   EOJ                  SALT MONITOR ENDS JOB
0002E0  CA00                                123+SVCEOJ   SVC   0
                                            124 *------------------------------------------------------------------------
                                            125 *        READ SVC - I/O NOT OVERLAPPED WITH PROBLEM PROG INST EXECUTION
0002F2  9400  33CE                003CE     126 SVCREAD  NI    SVCEXCP,X'00' DISALLOW EXCP
0002F6  D202  3311  3041  00311  00041      127          MVC   RDRCCW+1(3),OGPRS+5      READ INTO ADDR FROM GPR1
0002FC  41A0  3310                00310     128          LA    R10,RDRCCW
000300  50A0  3010                00010     129          ST    R10,CAW
000304  9C00  01C1                001C1     130          SIO   X'01C1'
000308  4770  3318                00318     131          BC    7,READERR
00030C  D207  308C  3024  0008C  00024      132 IORTRN   MVC   NIOPSW(8),OSVCPSW        RETURN TO USER AFTER INT
000312  D23F  309C  303C  0009C  0003C      133          MVC   NGPRS(64),OGPRS          WITH USER'S REGISTERS
000318  9602  3025                00025     134          OI    OSVCPSW+1,X'02'    PROB/WAIT STATES-INT'S ENABLED
00031C  8200  3024                00024     135          LPSW  OSVCPSW          WAIT ON I/O
000310  0200031000000050                    136 RDRCCW   CCW   READ,RDRCCW,NOFLAGS,80
000318  41A0  3290                00290     137 READERR  LA    R10,IOMSG
00031C  47F0  3110                00110     138          B     CANJOB
                                            139 *------------------------------------------------------------------------
                                            140 *        WRITE SVC -I/O NOT OVERLAPPED WITH PROGRAM INSTRUCTION EXEC
000320  9400  33CE                003CE     141 SVCWRITE NI    SVCEXCP,X'00'       DISALLOW EXCP
000324  41A0  3388                00388     142          LA    R10,PTRCCW
000328  50A0  3010                00010     143          ST    R10,CAW
00032C  58A0  3040                00040     144          L     R10,OGPRS+4
000330  9540  A000                00000     145          CLI   0(R10),C' '         TEST USER CARRIAGE CONTROL
000334  4780  3350                00350     146          BE    SPACES1
000338  95F0  A000                00000     147          CLI   0(R10),C'0'
00033C  4780  3358                00358     148          BE    SPACES2
000340  9560  A000                00000     149          CLI   0(R10),C'-'
000344  4780  3360                00360     150          BE    SPACES3
000348  95F1  A000                00000     151          CLI   0(R10),C'1'
00034C  4780  3368                00368     152          BE    SKIPGE
000350  920F  3388                00388     153 SPACES1  MVI   PTRCCW,X'0F'
000354  47F0  336C                0036C     154          B     DOIO
000358  920B  3388                00388     155 SPACES2  MVI   PTRCCW,X'0B'
```

```
                                                                    MINI SUPERVISOR-INTERRUPT HANDLING ROUTINES                    PAGE   4
  LOC     OBJECT CODE      ADDR1  ADDR2    STMT    SOURCE STATEMENT            VERSION 2 U OF A  SALT   4 DECEMBER 1970

00035C  47F0  336C                00036C    156             B     DOIO
000360  9207  3388                00388     157    SPACES3  MVI   PTRCCW,X'07'
000364  47F0  336C                0036C     158             B     DOIO
000368  9203  3388                00388     159    SKIPGE   MVI   PTRCCW,X'03'
00036C  41AA  0001                00001     160    DOIO     LA    R10,1(R10)          USER BUFFER + 1
000370  50A0  3000                00000     161             ST    R10,SUPER
000374  D202  3391  3001   00391  00001     162             MVC   PTRCCW+9(3),SUPER+1
00037A  9C00  02C2                002C2     163             SIO   X'02C2'
00037E  4770  3318                00318     164             BC    7,WRITERR
000382  47F0  32FC                002FC     165             B     IORTRN
000388  0000038840000000                    166    PTRCCW   CCW   X'00',PTRCCW,CCOM,0      CONTROL CCW
000390  0100038800000078                    167             CCW   WRITE,PTRCCW,NOFLAGS,120
000319                                      168    WRITERR  EQU   READERR
                                            169    *-----------------------------------------------------------------------
                                            170    *        CHAR TO INTEGER CONVERSION SVC
000398  5800  303C                0003C     171    SVCCTI   L     R0,OGPRS
00039C  5810  3040                00040     172             L     R1,OGPRS+4
0003A0  0A03                                173             SVC   3               CALL ON SALT MONITOR FOR CONVERSION
0003A2  5020  3044                00044     174             ST    R2,OGPRS+8      REFLECT RESULT TO USER
0003A6  8200  3024                00024     175             LPSW  OSVCPSW
                                            176    *-----------------------------------------------------------------------
                                            177    *        ITC CONVERSION SVC
0003AA  5800  303C                0003C     178    SVCITC   L     R0,OGPRS
0003AE  5810  3040                00040     179             L     R1,OGPRS+4
0003B2  0A04                                180             SVC   4               CALL SALT MONITOR FOR CONVERSION
0003B4  5020  3044                00044     181             ST    R2,OGPRS+8
0003B8  8200  3024                00024     182             LPSW  OSVCPSW
                                            183    *-----------------------------------------------------------------------
                                            184    *        DUMP SVC
0003BC  9E00  02C2                002C2     185    SVCDUMP  HIO   X'02C2'         READY PRINTER
0003C0  5800  303C                0003C     186             L     R0,OGPRS
0003C4  5810  3040                00040     187             L     R1,OGPRS+4
0003C8  0A05                                188             SVC   5               CALL SALT MONITOR FOR DUMP
0003CA  8200  3024                00024     189             LPSW  OSVCPSW
                                            190    *-----------------------------------------------------------------------
                                            191    *        EXECUTE CHANNEL PROGRAM SVC
0003CE  45E0  346C                0046C     192    SVCEXCP  BAL   R14,GETUNIT
0003D2  9101  1008                00008     193             TM    8(R1),X'01'   CCB BUSY?
0003D6  4710  3402                00402     194             BO    BSYCCB          YES
0003DA  9D00  A000                00000     195             TIO   0(R10)
0003DE  4780  343A                0043A     196             BC    8,SNEWIO
0003E2  4760  341F                0041F     197             BC    6,QNEWIO
0003E6  41A0  33FE                003FE     198             LA    R10,BADIOMSG  FATAL I/O CONDITION
0003EA  47F0  3110                00110     199             B     CANJOB
0003FE  C9D5E5C1D3C9C4                      200    BADIOMSG DC    C'INVALID IO CONDITION'
000402  41A0  340A                0040A     201    BSYCCB   LA    R10,BSYCCBM
000406  47F0  3110                00110     202             B     CANJOB
00040A  C9D640E3D9E840                      203    BSYCCBM  DC    C'IO TRY ON BUSY CCB  '
00041E  41A0  3426                00426     204    QNEWIO   LA    R10,BSYUNITM  COULD QUEUE IO AT THIS POINT
000422  47F0  3110                00110     205             B     CANJOB
000426  C9D640E3D9E840                      206    BSYUNITM DC    C'IO TRY ON BUSY UNIT '
00043A  5821  0000                00000     207    SNEWIO   L     R2,0(R1)        GET CCW ADDR
00043E  5020  3010                00010     208             ST    R2,CAW          POINT CAW AT CCW
000442  9601  1008                00008     209             OI    8(R1),X'01'     MARK CCB AS BUSY
```

```
  LOC    OBJECT CODE      ADDR1  ADDR2    STMT   SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

000446  59A0 3520               00520    210            C      R10,PIOB
00044A  4770 3456               00456    211            BNE    *+12
00044E  5010 3524               00524    212            ST     R1,RIOB+4      POINT RDR I/O BLOCK AT CCB
000452  47F0 345A               0045A    213            B      *+8
000456  5010 352C               0052C    214            ST     R1,PIOB+4      POINT PTR I/O BLOCK AT CCB
00045A  9C00 A000               00000    215            SIO    0(R10)         INITIATE I/O
00045E  4780 3468               00468    216            BC     8,*+10         BRANCH IF SUCCESSFUL
000462  D201 1008 3018   00008  00018    217            MVC    8(2,R1),CSW+4 REFLECT CSW STATUS TO USER CCB
000468  8200 3024               00024    218            LPSW   OSVCPSW
                                         219   *
00046C  5810 3040               00040    220   GETUNIT  L      R1,OGPRS+4
000470  1B22                             221            SR     R2,R2
000472  5020 3000               00000    222            ST     R2,SUPER
000476  D201 3002 1004   00002  00004    223            MVC    SUPER+2(2),4(R1)   GET UNIT ADDR FROM CCB
00047C  58A0 3000               00000    224            L      R10,SUPER
000480  07FE                             225            BR     R14
                                         226   *-----------------------------------------------------------------------
                                         227   *        WAIT SVC      .
000482  45E0 346C               0046C    228   SVCWAIT  BAL    R14,GETUNIT
000486  9D00 A000               00000    229            TIO    0(R10)        TEST UNIT AND ITS STATUS
00048A  4740 3492               00492    230            BC     4,*+8         VALID UNIT IS BUSY
00048E  8200 3024               00024    231            LPSW   OSVCPSW       NOT BUSY OR INVALID-RETURN
000492  59A0 3520               00520    232            C      R10,RIOB      READER?
000496  4770 34A2               004A2    233            BNE    *+12          NO
00049A  9680 3524               00524    234            OI     RIOB+4,X'80'  MARK READER AS WAITED ON
00049E  47F0 34A6               004A6    235            B      *+8
0004A2  9680 352C               0052C    236            OI     PIOB+4,X'80'  MARK PTR AS WAITED ON
0004A6  94FE 1008               00008    237            NI     8(R1),X'FE'   BUSY FLAG OFF IN CCB
0004AA  9602 3025               00025    238            OI     OSVCPSW+1,X'02'    SET WAIT STATE
0004AE  8200 3024               00024    239            LPSW   OSVCPSW       PRESERVE STATUS - EXCEPT WAIT FIELD
                                         240   *-----------------------------------------------------------------------
0004B2  41A0 34BA               004BA    241   BADSVC   LA     R10,NOSVC
0004B6  47F0 3110               00110    242            B      CANJOB
0004BA  C9D5E5C1D3C9C4                   243   NOSVC    DC     C'INVALID SVC CODE    '
                                         244   *-----------------------------------------------------------------------
                                         245   ***   I/O INTERRUPT HANDLING ROUTINE
0004CE  1B22                             246   IOPROC   SR     R2,R2
0004D0  5020 3000               00000    247            ST     R2,SUPER
0004D4  D201 3002 302E   00002  0002F    248            MVC    SUPER+2(2),OIOPSW+2     GET UNIT ADDR FROM INT CODE
0004DA  58A0 3000               00000    249            L      R10,SUPER
0004DE  59A0 3520               00520    250            C      R10,RIOB      READER ADDRESSED
0004E2  4770 34EE               004EE    251            BNE    *+12
0004E6  4120 3520               00520    252            LA     R2,RIOB
0004EA  47F0 34F2               004F2    253            B      *+8
0004EE  4120 3528               00528    254            LA     R2,PIOB
0004F2  5812 0004               00004    255            L      R1,4(R2)      GET CCB ADDR
0004F6  D201 1006 301A   00006  0001A    256            MVC    6(2,R1),CSW+6      REFLECT RESIDUAL COUNT TO CCB
0004FC  D201 1008 3018   00008  00018    257            MVC    8(2,R1),CSW+4      REFLECT STATUS TO CCB
000502  9102 302D               0002D    258            TM     OIOPSW+1,X'02'     WAS CPU IN WAIT STATE?
000506  4710 350E               0050E    259            BO     *+8           YES
00050A  8200 302C               0002C    260   IOUT     LPSW   OIOPSW        CONTINUE PROBLEM PROG
00050E  1211                             261            LTR    R1,R1         WAITING ON THIS IO?
000510  4720 350A               0050A    262            BP     IOUT          NO-RETURN TO WAIT
000514  94FD 302D               0002D    263            NI     OIOPSW+1,X'FD'     TURN WAIT BIT OFF
```

```
                                        MINI SUPERVISOR-INTERRUPT HANDLING ROUTINES                PAGE   6
  LOC    OBJECT CODE      ADDR1  ADDR2    STMT   SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

000518  940C  2C04              00004     264            NI    4(R2),X'00'   REMOVE FLAG FROM I/O BLOCK
00051C  82C0  302C              0002C     265            LPSW  OIOPSW        RETURN TO PROBLEM PROG
                                          266  *
000520                                    267  RIOB      DS    OF
000520  C0C001C1                          268            DC    X'C0C001C1'   READER ADDR
000524  00000000                          269            DC    A(0)          CURRENT CCB ADDR
000528                                    270  PIOB      DS    OF
000528  000002C2                          271            DC    X'000002C2'   PRINTER ADDR
00052C  00000000                          272            DC    A(0)          CURRENT CCB ADDR
000530                                    273            DROP  3
000530                                    274  USENTRY   DS    OF
                                          275  ***   END OF SUPERVISOR
                                          276  *------------------------------------------------------------------ ------
```

```
                                                       PROBLEM PROGRAM - TRY VARIOUS CCW S                       PAGE    7
  LOC    OBJECT CODE      ADDR1  ADDR2    STMT    SOURCE STATEMENT          VERSION 2 U OF A  SALT    4 DECEMBER 1970

000530  05A0                               278            BALR  B10,0
000532                                     279            USING *,B10
                                           280   *-----------------------------------------------------------------------
                                           281   *      SIMPLE CCW READ FOLLOWED BY SKIP PAGE CNTRLCCW CHAINED
                                           282   *      TO A SIMPLE CCW WRITE
                                           283            EXCP  RDRCCB1
000532  5810  A5F2                00B24    284 +          L     1,=A(RDRCCB1)
000536  0A06                               285 +          SVC   6
                                           286            WAIT  RDRCCB1
000538  5810  A5F2                00B24    287 +          L     1,=A(RDRCCB1)
00053C  9131  1008                00008    288 +          TM    8(1),X'31'
000540  4780  A014                00546    289 +          BZ    *+6
000544  0A07                               290 +          SVC   7
000546  9108  A19A                006CC    291            TM    RDRCCB1+8,X'08'     END OF FILE?
00054A  4710  A15E                00690    292            BO    ENDPROG
00054E  D24F  A3E6  A396   00918  008C8    293            MVC   LINE(80),CARD
                                           294            EXCP  PTRCCB1
000554  5810  A5F6                00B28    295 +          L     1,=A(PTRCCB1)
000558  0A06                               296 +          SVC   6
                                           297            WAIT  PTRCCB1
00055A  5810  A5F6                00B28    298 +          L     1,=A(PTRCCB1)
00055E  9131  1008                00008    299 +          TM    8(1),X'31'
000562  4780  A036                00568    300 +          BZ    *+6
000566  0A07                               301 +          SVC   7
                                           302   *-----------------------------------------------------------------------
                                           303   *      UNCHAINED PRINTER CONTROL CCW FOLLOWED BY A SIMPLE WRITE CCW
000568  D24F  A3E6  A53A   00918  00A6C    304            MVC   LINE(80),MSG1
                                           305            EXCP  PTRCCB2
00056E  5810  A5FA                00B2C    306 +          L     1,=A(PTRCCB2)
000572  0A06                               307 +          SVC   6
                                           308            WAIT  PTRCCB2
000574  5810  A5FA                00B2C    309 +          L     1,=A(PTRCCB2)
000578  9131  1008                00008    310 +          TM    8(1),X'31'
00057C  4780  A050                00582    311 +          BZ    *+6
000580  0A07                               312 +          SVC   7
                                           313            EXCP  PTRCCB3
000582  5810  A5FE                00B30    314 +          L     1,=A(PTRCCB3)
000586  0A06                               315 +          SVC   6
                                           316            WAIT  PTRCCB3
000588  5810  A5FE                00B30    317 +          L     1,=A(PTRCCB3)
00058C  9131  1008                00008    318 +          TM    8(1),X'31'
000590  4780  A064                00596    319 +          BZ    *+6
000594  0A07                               320 +          SVC   7
                                           321   *-----------------------------------------------------------------------
                                           322   *      SIMPLE WRITE WITH NO PRINTER CNTRL CAUSING PRINTER OVERSCORE
000596  D24F  A3E6  A588   00918  00ABA    323            MVC   LINE(80),MSG2
                                           324            EXCP  PTRCCB3
00059C  5810  A5FE                00B30    325 +          L     1,=A(PTRCCB3)
0005A0  0A06                               326 +          SVC   6
                                           327            WAIT  PTRCCB3
0005A2  5810  A5FE                00B30    328 +          L     1,=A(PTRCCB3)
0005A6  9131  1008                00008    329 +          TM    8(1),X'31'
0005AA  4780  A07E                005B0    330 +          BZ    *+6
0005AE  0A07                               331 +          SVC   7
```

```
                                          PROBLEM PROGRAM - TRY VARIOUS CCW S                                    PAGE   8
  LOC     OBJECT CODE       ADDR1  ADDR2   STMT   SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

                                            332  *-----------------------------------------------------------------------
                                            333  *      CHAIN DATA IN THE FORM OF A SCATTER READ FOLLOWED BY A SPACE 3
                                            334  *      CONTROL CCW COMMAND CHAINED TO A SIMPLE WRITE CCW
0005B0  9240  A3E6                 00918    335           MVI   LINE,C' '
0005B4  D276  A3E7  A3E6   00919   00918    336           MVC   LINE+1(119),LINE   BLANK LINE
                                            337           EXCP  RDRCCB2
0005BA  5810  A602                 00B34    338 +         L     1,=A(RDRCCB2)
0005BE  0AC6                                339 +         SVC   6
                                            340           WAIT  RDRCCB2
0005C0  5810  A602                 00B34    341 +         L     1,=A(RDRCCB2)
0005C4  9131  1008                 00008    342 +         TM    8(1),X'31'
0005C8  4780  A09C                 005CE    343 +         BZ    *+6
0005CC  0A07                                344 +         SVC   7
0005CE  9108  A1BA                 006EC    345           TM    RDRCCB2+8,X'08'     EOF?
0005D2  4710  A15E                 00690    346           BO    ENDPROG             YES
                                            347           EXCP  PTRCCB5
0005D6  5810  A606                 00B38    348 +         L     1,=A(PTRCCB5)
0005DA  0AC6                                349 +         SVC   6
                                            350           WAIT  PTRCCB5
0005DC  5810  A606                 00B38    351 +         L     1,=A(PTRCCB5)
0005E0  9131  1008                 00008    352 +         TM    8(1),X'31'
0005E4  4780  AC38                 005EA    353 +         BZ    *+6
0005E8  0A07                                354 +         SVC   7
                                            355  *-----------------------------------------------------------------------
                                            356  *      A READ CHANNEL PROGRAM USING THE SKIP DATA OPTION FOLLOWED BY A
                                            357  *      WRITE CHANNEL PROGRAM USING SKIP DATA
0005EA  9240  A396                 008C8    358           MVI   CARD,C' '
0005EE  D24E  A397  A396   008C9   008C8    359           MVC   CARD+1(79),CARD    BLANK INPUT BUFFER
0005F4  D213  A3AA  A60A   008DC   00B3C    360           MVC   CARD+20(20),=C'***NOT OVERWRITTEN**'
                                            361           EXCP  RDRCCB3
0005FA  5810  A61E                 00B50    362 +         L     1,=A(RDRCCB3)
0005FE  0A06                                363 +         SVC   6
                                            364           WAIT  RDRCCB3
000600  5810  A61E                 00B50    365 +         L     1,=A(RDRCCB3)
000604  9131  1008                 00008    366 +         TM    8(1),X'31'
000608  4780  AC0C                 0060E    367 +         BZ    *+6
00060C  0AC7                                368 +         SVC   7
                                            369           DUMP  CARD,80         ISSUE DUMP ONLY WHEN PRINTER FREE
00060E  4110  A396                 008C8    370 +         LA    1,CARD
000612  4100  0050                 00050    371 +         LA    0,80
000616  0AC5                                372 +         SVC   5
000618  92E7  A3E6                 00918    373           MVI   LINE,C'X'
00061C  D276  A3E7  A3E6   00919   00918    374           MVC   LINE+1(119),LINE   FILL LINE WITH X'S
                                            375           EXCP  PTRCCB6
000622  5810  A622                 00B54    376 +         L     1,=A(PTRCCB6)
000626  0AC6                                377 +         SVC   6
                                            378           WAIT  PTRCCB6
000628  5810  A622                 00B54    379 +         L     1,=A(PTRCCB6)
00062C  9131  1008                 00008    380 +         TM    8(1),X'31'
000630  4780  A1C4                 00636    381 +         BZ    *+6
000634  0A07                                382 +         SVC   7
                                            383  *-----------------------------------------------------------------------
                                            384  *      CPU/CHANNEL OVERLAP DEMONSTRATED BY INITIATING CHANNEL TO
                                            385  *      WRITE FROM A BUFFER FILLED WITH A'S WHILE CPU INSTRUCTION
```

```
                                              PROBLEM PROGRAM - TRY VARIOUS CCW S                                    PAGE   9
  LOC     OBJECT CODE      ADDR1  ADDR2   STMT    SOURCE STATEMENT           VERSION 2 U OF A  SALT   4 DECEMBER 1970

                                           386  *       EXECUTION PROGRESSIVELY FILLS BUFFER FROM HIGH END WITH B'S
                                           387            EXCP   PTRCCB4
000636  5810  A626              00B58      388 +          L      1,=A(PTRCCB4)
00063A  0A06                               389 +          SVC    6
00063C  4150  C077              00077      390            LA     R5,119
000640  4140  A45E              00990      391  DESTROY   LA     R4,LINEA
000644  1A45                               392            AR     R4,R5
000646  D200  4000  A62A  00000 00B5C      393            MVC    0(1,R4),=C'B'
00064C  4650  A10E              00640      394            BCT    R5,DESTROY
                                           395            WAIT   PTRCCB4
000650  5810  A626              00B58      396 +          L      1,=A(PTRCCB4)
000654  9131  1008              00008      397 +          TM     8(1),X'31'
000658  4780  A12C              0065E      398 +          BZ     *+6
00065C  0A07                               399 +          SVC    7
                                           400  *-----------------------------------------------------------------------
                                           401  *       SOPHISTICATED CHANNEL PROGRAM EMPLOYING TRANSFER IN CHANNEL
                                           402  *       CONCURRENT WITH A READ THAT READS NO DATA BUT CAUSES END OF FILE
                                           403            EXCP   PTRCCB7
00065E  5810  A62E              00B60      404 +          L      1,=A(PTRCCB7)
000662  0A06                               405 +          SVC    6
                                           406            EXCP   RDRCCB1
000664  5810  A5F2              00B24      407 +          L      1,=A(RDRCCB1)
000668  0A06                               408 +          SVC    6
                                           409            WAIT   RDRCCB1
00066A  5810  A5F2              00B24      410 +          L      1,=A(RDRCCB1)
00066E  9131  1008              00008      411 +          TM     8(1),X'31'
000672  4780  A146              00678      412 +          BZ     *+6
000676  0A07                               413 +          SVC    7
                                           414            WAIT   PTRCCB7
000678  5810  A62E              00B60      415 +          L      1,=A(PTRCCB7)
00067C  9131  1008              00008      416 +          TM     8(1),X'31'
000680  4780  A154              00686      417 +          BZ     *+6
000684  0A07                               418 +          SVC    7
000686  9108  A19A              006CC      419            TM     RDRCCB1+8,X'08'     EOF?
00068A  4710  A15E              00690      420            BO     ENDPROG              YES
00068E  0000                               421            DC     X'0000'       NO-CAUSE SALT INTERRUPT
                                           422  *-----------------------------------------------------------------------
                                           423  ENDPROG   DUMP   EOFMSG,ENDCCBS-EOFMSG
000690  4110  A16A              0069C      424 +ENDPROG   LA     1,EOFMSG
000694  4100  0160              00160      425 +          LA     0,ENDCCBS-EOFMSG
000698  0A05                               426 +          SVC    5
                                           427            EOJ
00069A  0A00                               428 +          SVC    0


                                           430  *-----------------------------------------------------------------------
                                           431  *       CHANNEL CONTROL BLOCKS WITH DUMP FORMAT MESSAGES
                                           432  *       READER
00069C                                     433            DS     0F
00069C  5C5CC3D6D5E3C5                     434  EOFMSG    DC     C'**CONTENTS OF CCB S AT RDR EOF**'
0006BC  D9C4D9C3C3C2F1                     435            DC     C'RDRCCB1 '
                                           436  RDRCCB1   CCB    X'01C1',RDRCCW1          SIMPLE READ
0006C4  000007FC                           437 +RDRCCB1   DC     A(RDRCCW1)
```

```
                                                    PROBLEM PROGRAM - TRY VARIOUS CCW S                         PAGE  10
  LOC    OBJECT CODE      ADDR1 ADDR2   STMT    SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

0006C8  01C1                             438 +           DC    X'01C1'
0006CA  0000                             439 +           DC    2X'00'
0006CC  0000                             440 +           DC    2X'00'
0006CE  40404040404040                   441             DC    14C' '
0006DC  D9C4D9C3C3C2F2                   442             DC    C'RDRCCB2 '
                                         443  RDRCCB2    CCB   X'01C1',RDRCCW2          SCATTER READ
0006E4  00000804                         444 +RDRCCB2    DC    A(RDRCCW2)
0006E8  01C1                             445 +           DC    X'01C1'
0006EA  0000                             446 +           DC    2X'00'
0006EC  0000                             447 +           DC    2X'00'
0006EE  40404040404040                   448             DC    14C' '
0006FC  D9C4D9C3C3C2F3                   449             DC    C'RDRCCB3 '
                                         450  RDRCCB3    CCB   X'01C1',RDRCCW3          SKIP DATA READ
000704  0000081C                         451 +RDRCCB3    DC    A(RDRCCW3)
000708  01C1                             452 +           DC    X'01C1'
00070A  0000                             453 +           DC    2X'00'
00070C  0000                             454 +           DC    2X'00'
00070E  40404040404040                   455             DC    14C' '
                                         456  *       PRINTER              .
00071C  D7E3D9C3C3C2F1                   457             DC    C'PTRCCB1 '
                                         458  PTRCCB1    CCB   X'02C2',PTRCCW1      SKIP PAGE WITH SIMPLE WRITE
000724  00000834                         459 +PTRCCB1    DC    A(PTRCCW1)
000728  02C2                             460 +           DC    X'02C2'
00072A  0000                             461 +           DC    2X'00'
00072C  0000                             462 +           DC    2X'00'
00072E  40404040404040                   463             DC    14C' '
00073C  D7E3D9C3C3C2F2                   464             DC    C'PTRCCB2 '
                                         465  PTRCCB2    CCB   X'02C2',PTRCCW2    SINGLE SPACE 2 CONTROL CCW
000744  00000844                         466 +PTRCCB2    DC    A(PTRCCW2)
000748  02C2                             467 +           DC    X'02C2'
00074A  0000                             468 +           DC    2X'00'
00074C  0000                             469 +           DC    2X'00'
00074E  40404040404040                   470             DC    14C' '
00075C  D7E3D9C3C3C2F3                   471             DC    C'PTRCCB3 '
                                         472  PTRCCB3    CCB   X'02C2',PTRCCW3    SINGLE WRITE CCW
000764  00000830                         473 +PTRCCB3    DC    A(PTRCCW3)
000768  02C2                             474 +           DC    X'02C2'
00076A  0000                             475 +           DC    2X'00'
00076C  0000                             476 +           DC    2X'00'
00076E  40404040404040                   477             DC    14C' '
00077C  D7E3D9C3C3C2F4                   478             DC    C'PTRCCB4 '
                                         479  PTRCCB4    CCB   X'02C2',PTRCCW4          SPACE 1 CNTRL WITH SIMPLE WRITE
000784  0000084C                         480 +PTRCCB4    DC    A(PTRCCW4)
000788  02C2                             481 +           DC    X'02C2'
00078A  0000                             482 +           DC    2X'00'
00078C  0000                             483 +           DC    2X'00'
00078E  40404040404040                   484             DC    14C' '
00079C  D7E3D9C3C3C2F5                   485             DC    C'PTRCCB5 '
                                         486  PTRCCB5    CCB   X'02C2',PTRCCW5          SPACE 3 CNTRL WITH SIMPLE WRITE
0007A4  0000085C                         487 +PTRCCB5    DC    A(PTRCCW5)
0007A8  02C2                             488 +           DC    X'02C2'
0007AA  0000                             489 +           DC    2X'00'
0007AC  0000                             490 +           DC    2X'00'
0007AE  40404040404040                   491             DC    14C' '
```

```
                                              PROBLEM PROGRAM - TRY VARIOUS CCW S                            PAGE  11
  LOC     OBJECT CODE      ADDR1  ADDR2  STMT   SOURCE STATEMENT          VERSION 2 U OF A  SALT   4 DECEMBER 1970

0007BC  D7E3D9C3C3C2F6                   492            DC     C'PTRCCB6 '
                                         493   PTRCCB6  CCB    X'02C2',PTRCCW6          SPACE 1 WITH SKIP DATA WRITE
0007C4  0000086C                         494  +PTRCCB6  DC     A(PTRCCW6)
0007C8  0202                             495  +         DC     X'02C2'
0007CA  0000                             496  +         DC     2X'00'
0007CC  0000                             497  +         DC     2X'00'
0007CE  40404040404040                   498            DC     14C' '
0007DC  D7E3D9C3C3C2F7                   499            DC     C'PTRCCB7 '
                                         500   PTRCCB7  CCB    X'02C2',PTRCCW7          SPACE 2-TRANSFER IN CHANNEL
0007E4  00000894                         501  +PTRCCB7  DC     A(PTRCCW7)
0007E8  0202                             502  +         DC     X'02C2'
0007EA  0000                             503  +         DC     2X'00'
0007EC  0000                             504  +         DC     2X'00'
0007EE  40404040404040                   505            DC     14C' '
0007FC                                   506   ENDCCBS  EQU    *


                                         508  *-----------------------------------------------------------------------
                                         509  *        CHANNEL PROGRAMS
                                         510  *        READER
0007FC  020008C800000050                 511   RDRCCW1  CCW    READ,CARD,NOFLAGS,80      SIMPLE 80 BYTE READ
000804  0200092280000014                 512   RDRCCW2  CCW    READ,LINE+10,CDAT,20      CHAIN DATA CCW'S-
00080C  0200094080000028                 513            CCW    READ,LINE+40,CDAT,40      -FOR-
000814  0200097700000014                 514            CCW    READ,LINE+95,NOFLAGS,20  -SCATTER READ
00081C  020008C880000014                 515   RDRCCW3  CCW    READ,CARD,CDAT,20         READ FIRST 20 BYTES-
000824  020008DC90000014                 516            CCW    READ,CARD+20,CDATSK,20   -SKIP SECOND 20 BYTES-
00082C  020008F000000028                 517            CCW    READ,CARD+40,NOFLAGS,40  -READ LAST 40 BYTES
                                         518  *        PRINTER
000834  0300091840000000                 519   PTRCCW1  CCW    SKIPAGE,LINE,CCOM,0       SKIP PAGE AND-
00083C  0100091800000078                 520            CCW    WRITE,LINE,NOFLAGS,120   -WRITE LINE
000844  0B00091800000000                 521   PTRCCW2  CCW    SPACE2,LINE,NOFLAGS,0     SINGLE SPACE 2 CONTROL
00083C                                   522   PTRCCW3  EQU    PTRCCW1+8
00084C  0F00099040000000                 523   PTRCCW4  CCW    SPACE1,LINEA,CCOM,0       SPACE 1 AND-
000854  0100099000000078                 524            CCW    WRITE,LINEA,NOFLAGS,120  -WRITE LINE
00085C  0700091840000000                 525   PTRCCW5  CCW    SPACE3,LINE,CCOM,0        SPACE 3 AND-
000864  0100091800000078                 526            CCW    WRITE,LINE,NOFLAGS,120   -WRITE LINE
00086C  0F00091840000000                 527   PTRCCW6  CCW    SPACE1,LINE,CCOM,0        SPACE 1 AND-
000874  0100091880000014                 528            CCW    WRITE,LINE,CDAT,20       -WRITE 20 BYTES-
00087C  0100092C90000014                 529            CCW    WRITE,LINE+20,CDATSK,20 -SKIP 20 BYTES-
000884  0100094080000028                 530            CCW    WRITE,LINE+40,CDAT,40    -WRITE 40 BYTES-
00088C  0100096810000028                 531            CCW    WRITE,LINE+80,SKIP,40    -SKIP 40 BYTES
000894  0B00091840000000                 532   PTRCCW7  CCW    SPACE2,LINE,CCOM,0        SPACE 2 AND-
00089C  080008A840000000                 533            CCW    TIC,*+12,CCOM,0          -TRANSFER IN CHANNEL AND-
0008A4  00000000                         534            DC     F'0'
0008A8  0F00091840000000                 535            CCW    SPACE1,LINE,CCOM,0       -SPACE 1 AND-
0008B0  01000ABA80000014                 536            CCW    WRITE,MSG2,CDAT,20       -SCATTER WRITE 20 BYTES-
0008B8  01000A0840000064                 537            CCW    WRITE,BLANKS,CCOM,100    -WITH 100 BLANKS-
0008C0  0800086C40000000                 538            CCW    TIC,PTRCCW6,CCOM,0       -TRANSFER TO PREVIOUS CCW


                                         540  *-----------------------------------------------------------------------
                                         541  *        EQUATES, CONSTANTS AND BUFFERS
```

```
                                                         PROBLEM PROGRAM - TRY VARIOUS CCW S                                    PAGE  12
  LOC    OBJECT CODE        ADDR1  ADDR2     STMT     SOURCE STATEMENT           VERSION 2 U OF A  SALT    4 DECEMBER 1970

                                              542   *        CCW COMMANDS
000001                                        543   WRITE     EQU   1                  X'01'
000002                                        544   READ      EQU   2                  X'02'
000008                                        545   TIC       EQU   8                  X'08'      TRANSFER IN CHANNEL
00000F                                        546   SPACE1    EQU   15                 X'0F'
00000B                                        547   SPACE2    EQU   11                 X'0B'
000007                                        548   SPACE3    EQU   7                  X'07'
000003                                        549   SKIPAGE   EQU   3                  X'03'
                                              550   *         CCW FLAGS
000000                                        551   NOFLAGS   EQU   0
000010                                        552   SKIP      EQU   16                 X'10'      SKIP DATA
000040                                        553   CCOM      EQU   64                 X'40'      CHAIN COMMAND
000080                                        554   CDAT      EQU   128                X'80'      CHAIN DATA
000090                                        555   CDATSK    EQU   144                X'90'      CHAIN DATA AND SKIP DATA
                                              556   *        BUFFERS
0008C8  4040404040404040                      557   CARD      DC    80C' '
000918  4040404040404040                      558   LINE      DC    120C' '
000990  C1C1C1C1C1C1C1C1                      559   LINEA     DC    120C'A'
000A08  4040404040404040                      560   BLANKS    DC    100C' '
                                              561   *        MESSAGES AND MISC
000A6C  E3C8C9E240D3C9                        562   MSG1      DC    C'THIS LINE PRINTED BY UNCHAINED WRITE CCW PRECEDED BY '
000AA1  E4D5C3C8C1C9D5                        563             DC    C'UNCHAINED SPACE 2 CONTROL'
000ABA  E7E7E7E7E7E7E7E7                      564   MSG2      DC    20C'X'
000ACE  40E3C8C9E240D3                        565             DC    C' THIS LINE PRINTED WITH NO PREVIOUS PRINTER CONTROL'
000B01  4040404040404040                      566             DC    35C' '
000000                                        567   R0        EQU   0
000001                                        568   R1        EQU   1
000002                                        569   R2        EQU   2
000003                                        570   R3        EQU   3
000004                                        571   R4        EQU   4
000005                                        572   R5        EQU   5
00000A                                        573   B10       EQU   10
00000A                                        574   R10       EQU   10
00000E                                        575   R14       EQU   14
00000F                                        576   R15       EQU   15
                                              577   *---------------------------------------------------------------------------
000B24                                        578             END   DONE ASSEMBLY
000B24  C00556C4                              579                   =A(RDRCCB1)
000B28  00055724                              580                   =A(PTPCCB1)
000B2C  C0055744                              581                   =A(PTPCCB2)
000B30  00055764                              582                   =A(PTRCCB3)
000B34  C00556E4                              583                   =A(RDRCCB2)
000B38  C00557A4                              584                   =A(PTRCCB5)
000B3C  5C5C5CD5D6E340                        585                   =C'***NOT OVERWRITTEN***'
000B50  00055704                              586                   =A(RDRCCB3)
000B54  000557C4                              587                   =A(PTRCCB6)
000B58  C0055784                              588                   =A(PTRCCB4)
000B5C  C2                                    589                   =C'B'
000B60  C00557E4                              590                   =A(PTRCCB7)

                                                       0  STATEMENT(S) FLAGGED IN THIS ASSEMBLY
```

```
THIS CARD WAS READ BY RDRCCB1-MOVED TO LINE-WRITTEN BY PTRCCB1 WHICH SKIPS PAGE*

THIS LINE PRINTED BY UNCHAINED WRITE CCW PRECEDED BY UNCHAINED SPACE 2 CONTROLXX
***OVERSCORE***XXXXX THIS LINE PRINTED WITH NO PREVIOUS PRINTER CONTROL

          *20 BYTES TO LINE+10          *THE NEXT 40 BYTES INTO LINE+40XXXXXXXXX               *20 BYTES TO LINE+95

DUMP CALLED FROM 0003C8

RELOCATION FACTOR = 055000

REGS 0-7             0000005C   000558C8   00000000   00055000          00000000   00000000   00000000   00000000
REGS 8-15            00000000   00000000   00000014   0000000C          00C000C0   00000000   0000000C   000553BC

0008D8   E7E7E7E7 E7E7E7E7 E7E7E7E7 E7E7E7E7    E7E7E7E7 5C5C5CD5 D6E34CD6 E5C5D9E6   *XXXXXXXXXXXXXXXXXXX***NOT OVERW*
0008F8   D9C9E3E3 C5D55C5C E7E7E7E7 E7E7E7E7    E7E7E7E7 E7E7E7E7 E7E7E7E7 E7E7E7E7   *RITTEN**XXXXXXXXXXXXXXXXXXXXXXXX*
000918   E7E7E7E7 E7E7E7E7 E7E7E7E7 E7E7E7E7    40404040 40404040 40405CF2 F040C2E8   *XXXXXXXXXXXXXXX          *20 BY*


XXXXXXXXXXXXXXXXXXXX                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB


XXXXXXXXXXXXXXXXXXXX
*** VERSCORE***
XXXXXXXXXXXXXXXXXXXX                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

DUMP CALLED FROM 0003C8

RELOCATION FACTOR = 055000

REGS 0-7             0000016C   0005569C   00000000   00055000          00000000   00000C00   000C0000   C0000000
REGS 8-15            00000000   00000000   00000014   00000000          0000000C   00000000   00000000   000553BC

000698   0A05CACC 5C5CC3D6 D5E3C5D5 E3E240D6    C640C3C3 C240E240 C1E340D9 C4D940C5   *____**CONTENTS OF CCB S AT RDR E*
0006B8   D6C65C5C D9C4D9C3 C3C2F140 000557FC    01C10050 08004040 40404040 40404040   *OF**RDRCCB1 _____A_&__          *
0006D8   40404040 D9C4D9C3 C3C2F240 00055804    01C10000 C4004040 40404040 40404040   *    RDRCCB2 _____A____          *
0006F8   40404040 D9C4D9C3 C3C2F340 0005581C    01C10000 04004040 40404040 40404040   *    RDRCCB3 _____A____          *
000718   40404040 D7E3D9C3 C3C2F140 00055834    02C20000 00004040 40404040 40404040   *    PTRCCB1 _____B____          *
000738   40404040 D7E3D9C3 C3C2F240 00055844    02C20000 00004040 40404040 40404040   *    PTRCCB2 _____B____          *
000758   40404040 D7E3D9C3 C3C2F340 0005583C    02C20000 00004040 40404040 40404040   *    PTRCCB3 _____B____          *
000778   4040404C D7E3D9C3 C3C2F440 0005584C    02C20000 00004040 40404040 40404040   *    PTRCCB4 ___<_B____          *
000798   40404040 D7E3D9C3 C3C2F540 0005585C    02C20000 00004040 40404040 40404040   *    PTRCCB5 ___*_B____          *



EXECUTION TIME =     11270 MICRO-SECONDS ,  INSTRUCTIONS EXECUTED =     1409
CARDS READ =         3 ,  LINES PRINTED =         10
 CPU WAIT TIME =      9070 MICRO-SECONDS                                1409
      1 STUDENT JOBS IN BATCH
```